Anonymous Trust: Digital Rights Management Using Broadcast Encryption

JEFFREY LOTSPIECH, MEMBER, IEEE, STEFAN NUSSER, AND FLORIAN PESTONI

Invited Paper

Broadcast encryption is an active area of cryptographic research. Originally defined by Fiat and Naor, broadcast encryption refers to key management schemes that operate when the participating parties do not have a two-way communication path. We contrast that with public-key cryptography: all known public-key protocols require a handshake to establish a common key. We extend the use of broadcast encryption to solve problems that have been traditionally addressed by public-key cryptography: we discuss the xCP Cluster Protocol, a proposed digital rights management (DRM) system for the home entertainment network, and we illustrate a broadcast-encryption-based content distribution system, which can work without requiring any secrets in the DRM client.

Keywords—*Broadcast encryption, digital rights management (DRM), home network, xCP.*

I. Introduction

At their heart, digital rights management (DRM) systems are built on cryptography. However, it is not the us-versus-them cryptography reminiscent of a thousand spy movies. The attacker is not usually a nameless outsider trying to eavesdrop on a secret conversation between two parties who are both motivated to prevent the attack. Instead, the attacker is indistinguishable *a priori* from a legitimate user.

What is the attacker trying to do? Ideally, he or she wants to get the content for free, but it is relatively easy for the DRM system to prevent that. So often the attacker just wants to become a hero by liberating the content and making it widely available on the Internet, which is much harder for the system to prevent. In the latter case, the system unavoidably ends up *giving the attacker the key*.

Manuscript received July 15, 2003; revised February 9, 2004.

J. Lotspiech is with the IBM Almaden Research Center, Henderson, NV 89052 USA (e-mail: lotspiech@almaden.ibm.com).

S. Nusser and F. Pestoni are with the IBM Almaden Research Center, San Jose, CA 95120 USA (email: nusser@us.ibm.com; fpestoni@almaden.ibm.com).

Digital Object Identifier 10.1109/JPROC.2004.827353

Therefore, the use of cryptography in DRM systems is not primarily about excluding external attacks; it is about providing a hook to force compliance, i.e., a way to make sure the client software plays by the rules. The client ends up having secret keys, and software tamper-resistance mechanisms to hide those keys—even from the end user on whose behalf it operates. To a classic cryptographer, this whole thing seems a little strange. It is certainly not possible to prove security; in fact, since the attackers are given the keys, in a certain sense, all DRM systems are provably insecure.

Into this twilight world of DRM systems have come many practitioners equipped with the traditional tools of cryptography. We have seen many DRM approaches that we call "Crypto 101": both the client and the server have public-key certificates, digitally signed by a common certificate authority. The client and the server have a handshake protocol in which both sides prove to the other that they are in possession of the secret private keys that correspond to the public keys in their certificates. As a side effect of the handshake, the two sides agree upon a unique secret key to encrypt further communications. These protected communications contain payment data from the client to the server and the key to decrypt the purchased content from the server to the client.

Hidden in the handshake protocol should have been some test to see if either party has been revoked. Possession of a certificate alone should not be proof that the entity is legitimate; a *current* copy of the list of revoked devices is also necessary. Otherwise, the first lost private key compromises the entire system, and the expensive public-key mechanism degenerates into a global secret scheme. Alas, this critical aspect of public-key cryptography is neglected in some proposed DRM systems.

This dependency on an online handshake protocol makes public-key-based systems unsuitable for physical media or broadcast-based distribution. The handshake would have to be incorporated into the system solely to accommodate the DRM functionality. The handshake is undesirable, since it prevents disconnected operations and raises privacy concerns. In other words, a public-key-based DRM system does not lend itself well to disconnected distribution models.

Recent cryptographic advances have provided an alternative to the "Crypto 101" approach based on public keys. This area of cryptography is called *broadcast encryption*. We argue that broadcast encryption offers interesting, perhaps even compelling, advantages in implementing DRM systems.

II. Broadcast Encryption

In 1993, Fiat and M. Naor considered the question, "is it possible to have a key management scheme with revocation, but without the participating parties having a two-way handshake?" [1] The answer was yes, and they called their solution "broadcast" encryption to emphasize its one-way nature [13].

The first broadcast encryption schemes had a classic size/performance tradeoff compared to public-key cryptography: although no handshake was necessary, the data that was transferred in one direction in broadcast encryption was much larger than the amount of data that would have been transferred in a two-way public-key handshake. On the other hand, the calculations needed for broadcast encryption require orders of magnitude less time. In 2001, a team at IBM led by D. Naor invented a broadcast encryption scheme where the data transferred was of the same order as the size of a public-key certificate revocation list, without increasing (in fact, even reducing) the scheme's processing overhead [4].

A. Matrix-Based Schemes

Broadcast encryption has had a remarkably short path from initial theory to commercial success. The properties of broadcast encryption make it essential for protecting content on physical media. If a recorder makes a recording, you would like any player in the world be able to play it back years later. There could be no opportunity for the player and the recorder to have a handshake and communicate the encryption key, nor would it be desirable to have a central authority that both devices would have to connect to in order to obtain the key. Therefore, the encryption key has to be communicated with only a one-way path from the recorder to the player.

The protection scheme for DVD video, called the content scrambling system (CSS), was designed and deployed soon after broadcast encryption was invented, and no one working on the new technology was aware of that largely theoretical work. As a result, CSS used a shared secret scheme. In 1999, in a story that we believe to be at least 50% apocryphal, it was reported that a 16-year-old in Norway had found a shared secret and had broken CSS. Regardless of the actual facts of the break, it is inarguable that the CSS scheme has been broken. It is now easy to find on the Internet programs that will decrypt a DVD video disk and burn copies onto unprotected media or trade the file electronically; this is what we call a *clone* or a *circumvention device*. There is no way, short of redesigning the system from scratch and recalling millions of players, to fix the break.

Newer media types, however, use content protection schemes based on broadcast encryption [5]. The schemes used in DVD audio, DVD video recorders, secure digital memory cards, and secure CompactFlash all use broadcast encryption. The various schemes are called content protection for recordable media (CPRM) [7] or, in the case of DVD audio, content protection for prerecorded media (CPPM) [8]. As of this writing (early 2004), over 120 companies have licensed CPRM and/or CPPM, and they have obtained keys for over 200 *million* devices so far.

The CPRM schemes use a relatively simple version of broadcast encryption, a description of which is useful as an example. First, a block of data, called the *media key block*, is prerecorded on blank media at manufacturing time. Each recorder or player reads the media key block and processes it to yield a key, which is called the *media key*. Each device processes this media key block in a slightly different way, but they all get the same answer in the end. On the other hand, when a previously detected circumvention device attempts to process the media key block in the same way, it ends up with the wrong answer.

The CPRM/CPPM media key block is associated with a matrix of keys, called device keys. This key matrix was generated by the CPRM licensing agency. The matrix is always much taller than it is wide (e.g., in the DVD recordable case, 16 columns wide by roughly 2500 rows tall) and is preembossed in the lead-in area on the disk. The media key block is the encryption, over and over again, of the media key using each different device key. If you have a device key, and you know its position in the matrix, you can go to that position and decrypt the value you find there. The result will be the media key.

Each device is given 16 device keys from the licensing agency, one for each column in the matrix. In Fig. 1, the keys of a single device are shown as the dark rectangles in the matrix. Every device in the world has a different set of device keys. If you pick two devices at random, they might have one or two keys in common, but no two devices will have all 16 keys in common. Each device processes the media key block by taking one of its device keys and decrypting that position in the matrix. Which device key? At first, it does not matter. Any column will do. But once the system comes under attack, circumvention devices will appear that are using some of the keys in the matrix. Once detected, these circumvention devices can be revoked[1] by storing garbage data in the positions of the media key block corresponding to the compromised keys, which are shown as X's in the figure above. If an innocent device finds that one of its keys has been X'd out, it simply moves on to another column. Eventually, it will find a key that has not been compromised. Of course, it is

[1]We use here the term *revoke* because it is commonly used in content protection literature and in the industry, although in this context it may be more appropriate to say that the device is *excluded*. The exclusion is relative to a particular piece of media, and does not affect the ability of the player to play content protected with older media key blocks.

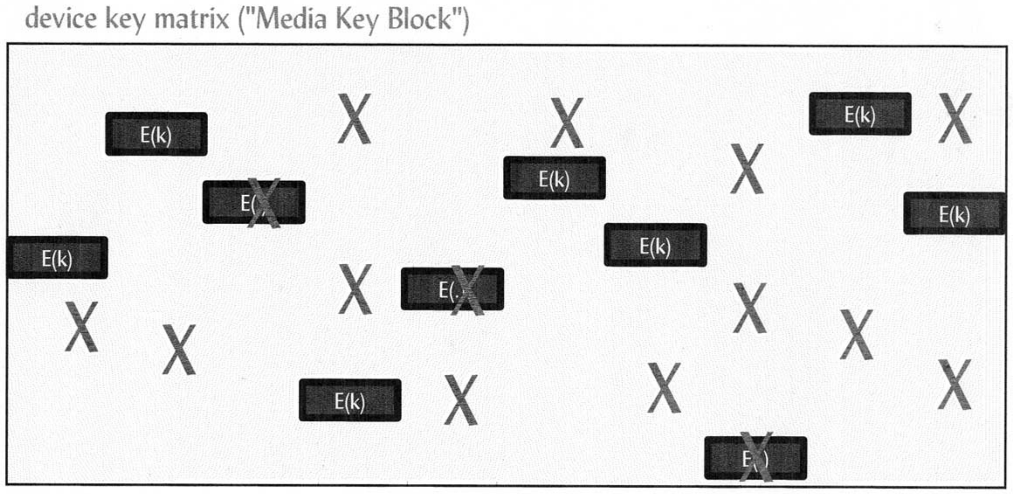

Fig. 1. CPRM key matrix.

possible that the system has been so compromised that most of the matrix has X's. In that case, some innocent devices would be excluded from the system, and we would probably say that the system has been broken.

B. Tree-Based Schemes

Since the days when CPRM was first introduced, there have been at least two new developments in broadcast encryption. First, researchers have invented superior tree-based schemes, and second, they have identified more applications than simple physical media protection. With the new applications, the term "media key block" has become inappropriate. The term "key management block" seems better, and we will use that when describing those other applications. Likewise, the "media key" becomes the "management key."

Matrix-based broadcast encryption schemes like CPRM have two drawbacks:

1) It is awkward and not completely satisfactory to increase the amount of total revocation capability (i.e., the size of the matrix) once the system has been deployed.
2) The system is especially sensitive to insider attacks—the so-called evil manufacturer problem, where a licensed manufacturer who misuses his keys can do damage equivalent to thousands of individual hacking events.

Starting in 1997, two independent groups developed a key management block idea based on trees—one group led by Wallner [2] and another led by Wong [3]. The two groups had the same idea.[2] This approach has come to be called the *logical key hierarchy* (LKH). LKH solved the drawbacks found in the matrix: because the size of an LKH key management block is based on the number of revocations, not the original size of the tree, it is straightforward to increase the amount of revocation capability in the field. Second, by grouping devices from a single manufacturer in the same place in the tree, it is easy to revoke the entire group of devices by lopping off that branch in tree. Thus, an evil manufacturer does no more damage than a single individual hacking event.

IBM's 2001 scheme completed the picture by designing a tree that was substantially more concise (25×) than an LKH tree [4]. This technology is called the subset-difference approach, or simply the NNL tree, after its authors, D. Naor, M. Naor, and Lotspiech. The size of the key management block in an NNL system is literally of the same order as the size of a public-key certificate revocation list, and this makes the NNL tree the only key management block system that is equivalent in revocation power to a public-key system. And, of course, public-key systems cannot be used to protect physical media because they require two-way conversations between the communicating parties.

To understand the NNL tree, it is helpful to first understand how the LKH tree works. In LKH, there is a large tree of keys. Each device is associated with a leaf of the tree. A binary tree with 4 billion nodes has 2 billion leaves, so such a tree could support 2 billion devices.[3] Each device is given the key associated with its leaf, and every other key associated with every other node in the tree on the path between the leaf and the root of the tree. (This corresponds to 32 keys per device in our example.) Now suppose you want to exclude a device; in other words, you want to give a management key to every other device in the world except the device you are excluding. All you need to do is encrypt the management key in the *sibling* keys of the keys that the circumvention device knows.

It is useful to look at an example. Suppose that Fig. 2 is an LKH tree, and we want to revoke the marked "device"—let us say it is device number 13 in the tree (i.e., it is the 13th leaf counting from 0 on the left of the tree). The device knows the

[2]Although neither group made the connection to content protection. That insight was made by Itkis [14].

[3]It is convenient to deal with balanced binary trees of 4 billion nodes, because each node (key) can identified with a single 32-b number, so in this discussion for concreteness we will assume all trees are this size.

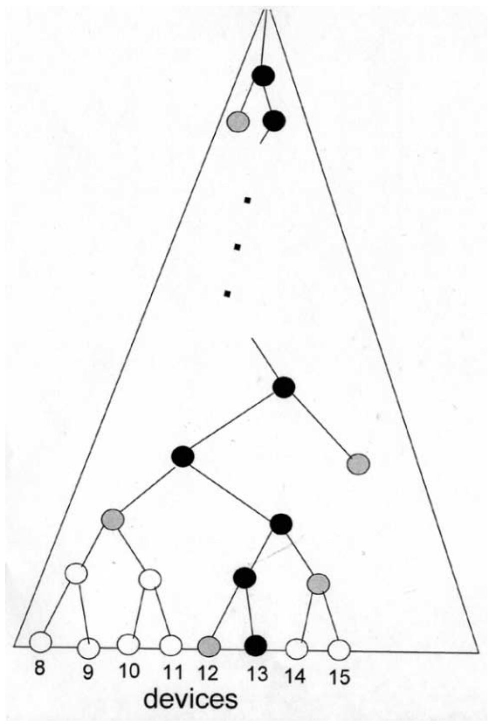

Fig. 2. LKH tree.

key for every black node in the figure—every key between it and the root of the tree. The sibling nodes are marked in gray. Look at device number 12, whose leaf is just to the left of the evil device. It has every key in common with the evil device, except for its leaf key. So to keep device 12 in the system, you have to encrypt the management key with its gray leaf key. Now look at devices 14 and 15 to the right of the evil device. They too have a lot of black nodes in common with the evil device, and, of course, you could just encrypt the management key with their leaf keys to keep them in the system. However, there is a more efficient way: the gray key that is one level higher is common to both 14 and 15, so you can keep them both in the system by just encrypting the management key using that key. Likewise, devices 8 through 11 can be kept in the system with just one encryption. With a little thought, you can convince yourself that every nonrevoked device in the system has exactly one gray (sibling) key. Using those keys would require 31 encryptions of the management key; that, and an indication of which number device is being revoked, is the fundamental information in an LKH key management block.

If there is more than one device that needs to be revoked, it is done in the same way, except now, when the paths from two revoked devices intersect, both sibling keys are black keys and there is nothing to encrypt at that level. So a key management block with more than one revocation has less than 31 encryptions per revocation, but not much less.

Now let us consider the NNL tree. You can think of it as the photographic negative of the LKH tree. In the LKH tree, the device has every key between its leaf and the root. In the NNL tree, the device has every key in the tree *except* the keys between its leaf and the root. So a device does not know its own leaf key, *but every other device in the system does*. It is easy to see why the NNL key management block is so concise: if you want to revoke device 13, just use device 13's leaf key to encrypt the management key; every other device knows it except device 13.

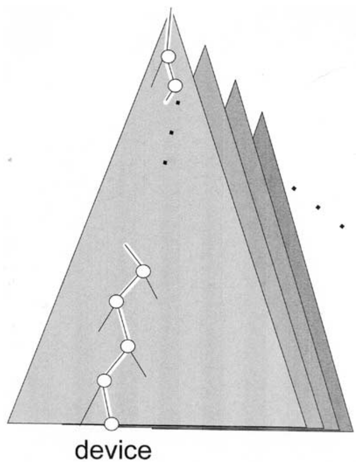

Fig. 3. NNL tree.

The trick happens when you want to revoke more than one device, let us say, device 13 and device 27. Device 13 knows device 27's leaf key, and vice versa, so which key can you use? To solve this problem, NNL does not just have a single system of keys; it has literally billions of systems of keys. In fact, every subtree in the master tree defines a completely independent system of keys. For example, one level down from the root of the tree, there are two subtrees, each with 1 billion devices in them. Each of these subtrees has its own system of keys, in addition to the system of keys in the master tree. Likewise, on the next level down, there are four device subtrees of size one-half billion devices, and each has an independent system of keys, and so on. Eventually, at the bottom, every pair of sibling leaf devices also defines its own system of keys. A device has a whole set of device keys in every subtree that it belongs to, 31 independent subtree systems per device. Fig. 3 illustrates the system of keys from a single device's point of view, with the subtrees shadowing off to the right. The nodes in white are the keys that the device does *not* know.

Now, with all these different systems based on subtrees, we have a mechanism to revoke all possible combinations of revoked devices. To go back to our example, we wanted to revoke both device 13 and device 27. The key management block says, "Use the system of keys for devices 0 : 15 only and use device 13's key in it, and use the system of keys for devices 16 : 31 only and use device 27's key in it." What happened to devices 32 to 2 billion? Here is another trick: the key management block also says, "Use the master system of keys for the entire tree, and use the key that revokes all devices 0 : 31 in it." Remember, all the nonrevoked devices between 0 and 31 have already learned the management key using their device keys in the smaller subtrees, so the fact that they are revoked in the larger one is inconsequential. This example gives you an idea of the number of encryptions that have to be stored in the NNL tree: there has to be one encryption per revocation, and occasionally there have to be other encryptions to pick up innocent devices not covered by the smaller subtrees. On average, there are 1.28 encryptions per revocation, as compared to the roughly 31 encryptions required in LKH. The ratio of the two numbers is the power of NNL.

It probably has not escaped the reader that the device has to know billions and billions of keys. This is true. How does it store them all? This is the last trick: the keys in the NNL trees and subtrees are not truly independent. Within a given subtree system, it is possible to calculate a key lower in the tree given a tree higher in the tree, because the lower level keys are picked to be secure one-way functions of the higher level keys. So, in fact, the device only has to store about 500 keys to know all of the keys in all of the systems it belongs to. However, the bigger subtrees are the most expensive in terms of device keys, and they are also the least likely to be used once the system has a reasonable number of revocations. So, in practice, the licensing agency can simply decide not to use the bigger subtrees in the key management block. Then, the number of keys the device needs to store can be cut in half or more. Without the large subtrees to fall back on, the licensing agency has to use slightly larger key management blocks in the case where there are few revocations, but that is usually a tradeoff gladly made.

Likewise, when we explained that each subtree was a completely independent system of keys, that was stretching the truth. They could be independent, but that would require the licensing agency to store billions of root keys for the billions of subtrees in the master tree. It does not weaken the scheme if there is some secret, cryptographically strong relationship between the root keys, known only to the licensing agency, and this can drastically reduce the number of keys that need to be stored in the agency.

There is one final optimization when you are trying to squeeze the last possible byte out of an NNL key management block. As described, the key management block contains the list of subtrees, which key should be used in each subtree, and the encryption of the management key in each of those keys. You cannot do anything about the encryptions, but it turns out that, given a certain revocation list, the optimal assignment of subtrees and keys within subtrees is unique, and the algorithm that calculates that optimal assignment is not too complicated. So it is possible to put the algorithm in each device and merely tell the device the list of revocations, asking the device to deduce the subtrees and keys within the subtrees. This optimization saves 2–3 b per revocation, and the extra cost in the device is negligible.

In summary, the NNL tree is the strongest known key management block technology in terms of number of revocations for a given size—at least an order of magnitude stronger than the next best alternative. In fact, it challenges public-key cryptography in that respect. However, unlike public-key cryptography, it can be used in applications like broadcast and media/storage protection, where there is no opportunity to have a handshake between the communicating parties. And it requires no complex, high-overhead calculations as are required in public-key systems.

III. Content Binding

In the previous section, we have shown different techniques to compute the management key or media key. In the CPRM application, this media key calculation is a necessary, but not sufficient condition to protect physical media. If the media key were used directly to encrypt the content on the media, then each piece of media would need a different media key block, to prevent users from being able to copy content by just making a bit-for-bit copy. On some types of physical media, such as DVDs, it is very important to be able to mass-produce preembossed media key blocks to achieve economies of scale. So in CPRM, cryptographic calculation for the unique media key K_{mu} is the cryptographic hash of the media key K_m with the unique media identifier ID_m

$$K_{\mathrm{mu}} = H(K_m, \mathrm{ID}_m). \tag{1}$$

In the case of DVD recordable disks, the media identifier is burned into the so-called burst cut area during the manufacturing of the blank disk. The burst cut area is not writable by normal DVD drives, so the media ID is a read-only value for all practical purposes.

The simple calculation in (1) is an example of what we call the *binding* step, and it is critical in using broadcast encryption to protect content. In general, we refer to this unique key as the *binding key*. Observe that the secret key K_{mu} can only be calculated by compliant devices, because only compliant devices can calculate K_m. But what if a user makes an unauthorized copy on to a different piece of media, but remembers the identifier of the old media? A device that possesses a set of device keys and knows the old media identifier *can* perform calculation (1) correctly and decrypt the content, but should refuse to. That is what it means to be compliant: the device plays by the rules. The implicit enforcement of this is that noncompliant devices—i.e., circumvention devices—could be excluded in new media key blocks. That is why we say content protection is fundamentally about compliance, not about cryptography. Cryptography is used as tool to set up a licensing structure.

CPRM has found it convenient to add a level of indirection into the content encryption. The media unique key does not directly encrypt the content; instead, it encrypts a title key K_t that in turn encrypts the content. If there are several pieces of content on a single piece of media, in general each content piece would have a separate title key. For each piece of content, the title key calculation is

$$D_i = e_{\mathrm{Kmu}}(K_{\mathrm{ti}} \oplus H[\mathrm{CCI}_i]) \tag{2}$$

where CCI is copy control information.

The encrypted value D_i is then stored on the media. Before it is encrypted, the title key is exclusive-ORed with a cryptographic hash of the content's CCI. The amount and scope of the CCI varies from content type to content type and media type to media type. It can be as simple as a single bit ("thou shalt not copy") or as involved as expiration dates or viewing counts. It is not encrypted; however, if a hacker were to modify the CCI, the title key calculation (2) would no longer be correct, and the content would not play. The CCI in CPRM is exactly analogous to the usage rights in a DRM system. It is not farfetched to imagine, therefore, that CPRM could be the basis of a DRM system.

IV. Server Side Binding—The Anonymous Trust System

We observe that the rise of CPRM devices has enabled a very interesting and relatively simple DRM system. A consumer would insert a blank recordable DVD into his PC. His client software would read the media key block and the media ID on the blank, and upload it to a DRM server, *without processing it*. The server would have a set of device keys to process the media key block, perform the binding calculation (1), and prepare a disk image of a movie, for example, bound to that particular piece of media. The client software would burn the DVD, and it would be playable in any DVD video player that supports CPRM. For example, since the DVD-RAM protection scheme is based on CPRM, any existing DVD player that supports the DVD-RAM standard could be used to play this DVD.

Consider the advantages of this hypothetical system, which we call an *anonymous trust* system. First, the client software contains no secrets, so it does not have to be tamper resistant, greatly simplifying its design and deployment. In fact, the client could even be open source. Second, one of the most problematic aspects of DRM design simply does not occur, namely, the question of when to charge the consumer for the download. If the server waits until there is a positive acknowledgment from the client that the content has been delivered before charging the consumer, a hacker may be able to get content for free by simply blocking that final step. And if the server charges the consumer at any earlier step, there is a chance that the consumer might be charged for content that was not actually delivered. In an anonymous trust system, however, the content has been customized to one particular piece of media, and can be downloaded over and over again without the extra downloads counting as extra copies. The server can simply keep downloading the disk image until the client gets it right. Note that this downloaded image will only play if written onto the particular blank media that was used when the transaction was initiated.

These advantages simplify the system, but there are definite advantages for the consumer as well. First, the content is designed to be consumed in the user's normal electronic devices like her TV and DVD player, not on her PC screen, which is awkward at best. Second, there is a concept in copyright law called the "doctrine of first sale," which states that royalties are only payable on the first sale of a piece of content; if the consumer later sells the copy she bought, she does not owe royalties on the sale. Many consumer advocate groups have interpreted this doctrine to mean the consumers should always have the *right* to sell anything they have bought electronically, and many DRM systems have found it very difficult to offer that function, especially without incurring extra transaction costs [6]. Setting aside the question of whether this is indeed a right, in an anonymous trust system it is supported automatically: the consumer ends up with a physical piece of media that he can, when he is done with it, give to a friend or sell on eBay.

Finally—and this is the "anonymous" part of anonymous trust—nothing in the actual delivery of content has identified the end user. It is much easier to maintain her privacy. At the same time, the content owners are confident that the content will not be misused, even if they do not know who they have given it to. If the content had been delivered to a circumvention device, the circumvention device would not have been able to process the media key block successfully and would not have been able to use the content. This is an important property of broadcast encryption: a source will have no idea who it is talking to, but it will have a cryptographic guarantee that the entity is a member in good standing of the same club.

Of course, the end user will have to pay for the content, and that might identify her. Certainly, a credit card transaction is not anonymous. But that is a separate problem, to which we will return later in this paper. For now, we will leave the existing world of CPRM to move on to what we see as a possible future use of broadcast encryption: content protection on a home entertainment network.

V. XCP Cluster Protocol and the Home Network

Over the last several years, the convergence of consumer electronics and computer technologies has accelerated. At the same time, broadband data connections and wireless, as well as wired technologies for home networking, have reached greater levels of market penetration. Next-generation entertainment devices are increasingly incorporating home networking technologies that allow easier access to content [9].

Home entertainment networks represent new opportunities for consumers and content owners, as well as new challenges for the protection of digital content. There is an inherent tension between the new technologies' ability to facilitate the flow of content across devices or nodes in the network and the content owners' requirement to control this flow. The resolution of this tension is again a matter of policy rather than technology, but any real-world implementation will in fact rely on a technological solution.

We present here a novel approach based on broadcast encryption for protecting content in a home network. To our knowledge, this is the only system specifically targeted at home networks that uses this technique; virtually all the other systems that we are aware of rely on public-key cryptography. As we have seen, there are several advantages in using broadcast encryption for content protection. It is somewhat surprising that you can use broadcast encryption in a peer-to-peer application.

Fig. 4 shows a typical home network. Note that some devices, like the family car, have wireless connections and may be only intermittently connected. Note also that there may be legacy devices connected to the network using existing copy-protected busses and media.

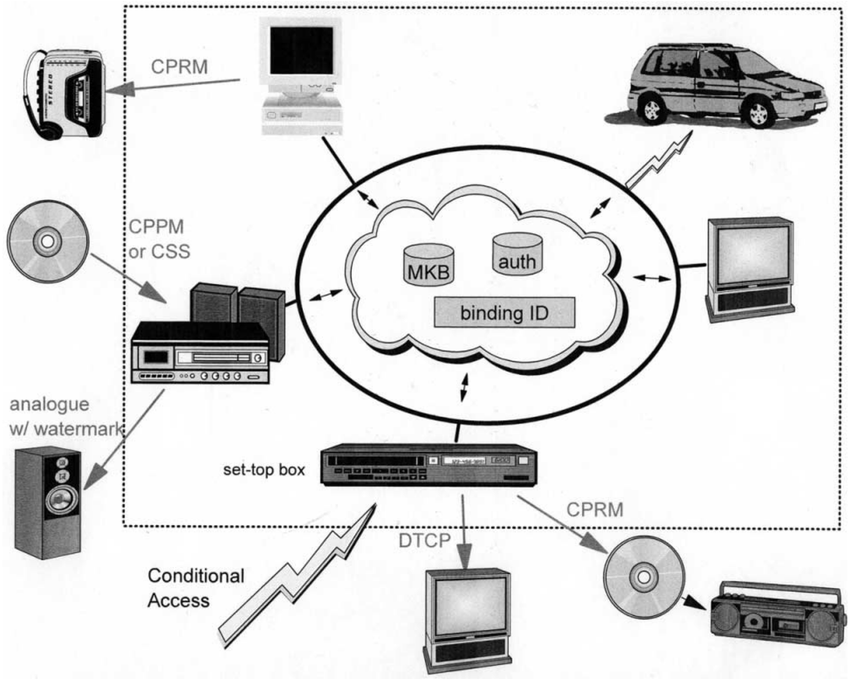

Fig. 4. Sample home network.

The system we propose is based on a peer-to-peer cryptographic protocol that allows a *cluster* of devices to agree on a common key for content encryption. The protocol guarantees that known circumvention devices cannot join a cluster with compliant devices in it, and that the notion of compliance can be updated to exclude previously unknown circumvention devices from further participation in any clusters they may have joined. When a device joins a cluster, it exchanges messages with the existing devices[4] that results in all devices sharing common cryptographic elements that allow them to access content protected for use in the specific cluster. However, only a limited number of devices can join the cluster, to prevent this home network from degrading into the "million device cluster" with devices connected over the Internet, which would clearly be in conflict with the requirements of content owners.

The devices in the xCP cluster have agreed upon three things: they have a common key management block; they have a common network identifier, called the *binding identifier*; and they have a common idea of what other devices are on the network, called the *authorization table*. These three things are bound together cryptographically: the management key from the key management block, the binding identifier, and a hash of the authorization table are used to calculate the common cluster key, the binding key

$$K_b = H\left(K_m, \mathrm{ID}_b \oplus H[\text{Auth table}]\right). \tag{3}$$

All content in the home is fundamentally protected by this binding key. As in CPRM, there is a level of indirection: the binding key encrypts the title keys for each piece of content, and the title keys are used to actually encrypt the content itself.

[4]The protocol also contemplates the case of bootstrapping a cluster with a "first" device.

Observe that after this initial exchange of data, because the protocol is based fundamentally on broadcast encryption, devices can calculate the binding key without having to have a conversation with any other device on the network. This is the big strength of xCP and contributes to its amazing flexibility regarding network transport: as long as the device can find the key management block and the authorization table (which are simple files in the network, duplicates of which might even be in the device's local persistent storage) and knows the binding ID, the device has everything it needs cryptographically. It can decrypt any piece of content in the network.

Of course, the usage rules that are cryptographically bound to that content may forbid it from doing certain things with the content. The device, because it is compliant, will not perform the forbidden action: for example, a recorder would not record something marked "do not copy." Moreover, to support greater flexibility, the default behavior for compliant devices is to refuse to access content that has associated usage rules that the device does not understand or cannot process.

Imagine that a user wants to make an unauthorized copy of some content for a friend. If he simply brings the copy over to his friend's house and loads it up on his friend's network, there is no chance it will play: the friend's network is using a different binding key, and her devices will not be able to calculate the title keys on this foreign content correctly.

But suppose the original user is more sophisticated, and brings along his network's key management block and his network's authorization table. After all, they are just simple files. Suppose he also knows his network's binding identifier (although this is not easy for him to determine). Still, this

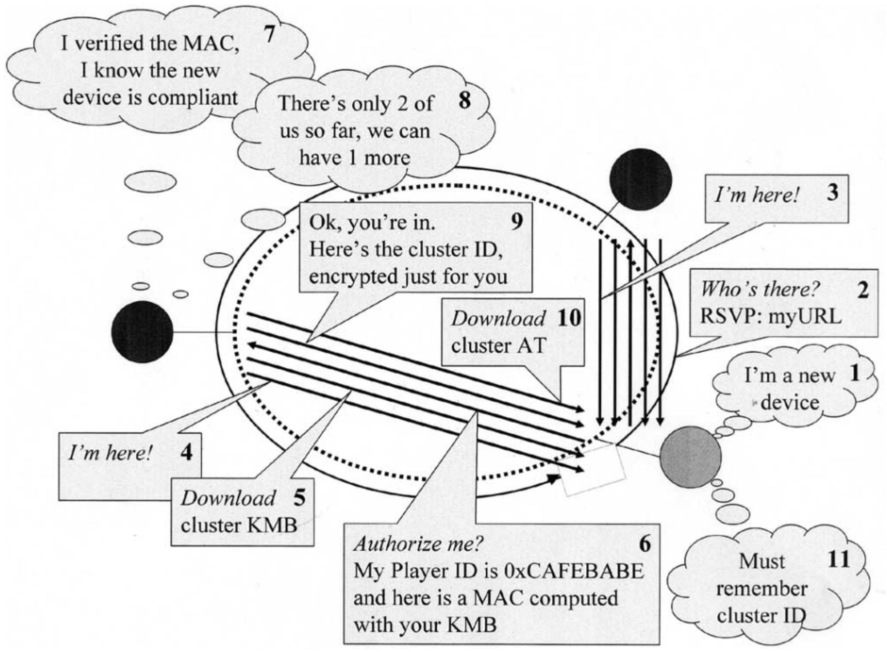
I verified the MAC, I know the new device is compliant 7
There's only 2 of us so far, we can have 1 more 8
Ok, you're in. Here's the cluster ID, encrypted just for you 9
I'm here! 3
Who's there? RSVP: myURL 2
Download cluster AT 10
I'm a new device 1
I'm here! 4
Download cluster KMB 5
Authorize me? My Player ID is 0xCAFEBABE and here is a MAC computed with your KMB 6
Must remember cluster ID 11

Fig. 5. Device join.

unauthorized content will not play. The compliant devices in his friend's network will observe that they are not on the authorization table and refuse to play, even though they could have calculated the binding key correctly.

Protocol Details: When a consumer purchases a device and connects it in his home, the device automatically sends a "Who's There?" broadcast message to determine which other xCP devices are currently in the home. Some of the devices will respond that they are "authorizers," and can authorize the new device. Some of the devices will respond that they are key management block servers (KMB servers), meaning they have a copy of the cluster's key management block and can share it. In practice, authorizers and KMB servers are usually the same devices—any device with storage will probably choose to be both an authorizer and a KMB server. (See Fig. 5.)

Now, the new device will ask *all* the authorizers to authorize it by sending an "authorize me" message to each one. In this message, the device identifies itself and its type, and "signs" the message with a message authentication code (MAC) based upon, fundamentally, the management key in the cluster's current key management block. By checking the MAC, the authorizer is confident that this new device is not a circumvention device. The authorizer would then check the authorization table and figure out whether accepting an additional device would not exceed the maximum number of devices allowed in the cluster.[5]

The new device is authorized if any one of the authorizers says so. It is perfectly normal for one authorizer to authorize the new device and one to reject it; this happens, for example, if the cluster is large, and only one of them has the necessary connection to an external authorization center. In Fig. 5, though, we assume that both authorizers have authorized this new device. In the response, they tell the device this network's binding ID (encrypted in a key based on the management key). The authorizers also then talk to each other, notifying each other that there is a new network binding key.

Remember that the authorization table is part of the calculation of the binding key and will have changed when the new device joined. A side effect of this is that content must be reencrypted; fortunately, because of the level of indirection mentioned above, we only need to reencrypt the title keys, each of which are comparatively short.

Suppose this new device also has storage and is prepared to become another authorizer and KMB server on the network. Such a device will have come preinstalled with its own key management block. It does not want to just blindly accept the key management block that is currently in use. It might be an old, cracked key management block and these existing devices might be a group of circumvention devices who are just waiting for a fresh compliant device to come into their midst to start obtaining new content. The new compliant device would like its key management block to become the new network key management block.

The existing devices have a slightly more subtle concern: they know this new device is not a circumvention device, because it has proven that it can process the current network key management block. But they are worried that it might be an old compliant device with an old, cracked key management block. A tricky user could keep this old device off to the side, and reintroduce it every time he wants the key management block to regress. Fortunately, with the tree-style key management blocks, it is possible for devices to determine whether one key management block is a subset of another key management block (in terms of revocation). The rule is that devices always agree upon the key management block that is the superset. If the two are disjoint, the devices merge

[5]The protocol also includes an exception mechanism for clusters that are larger than the predefined maximum. However, a description of this mechanism is outside the scope of this paper.

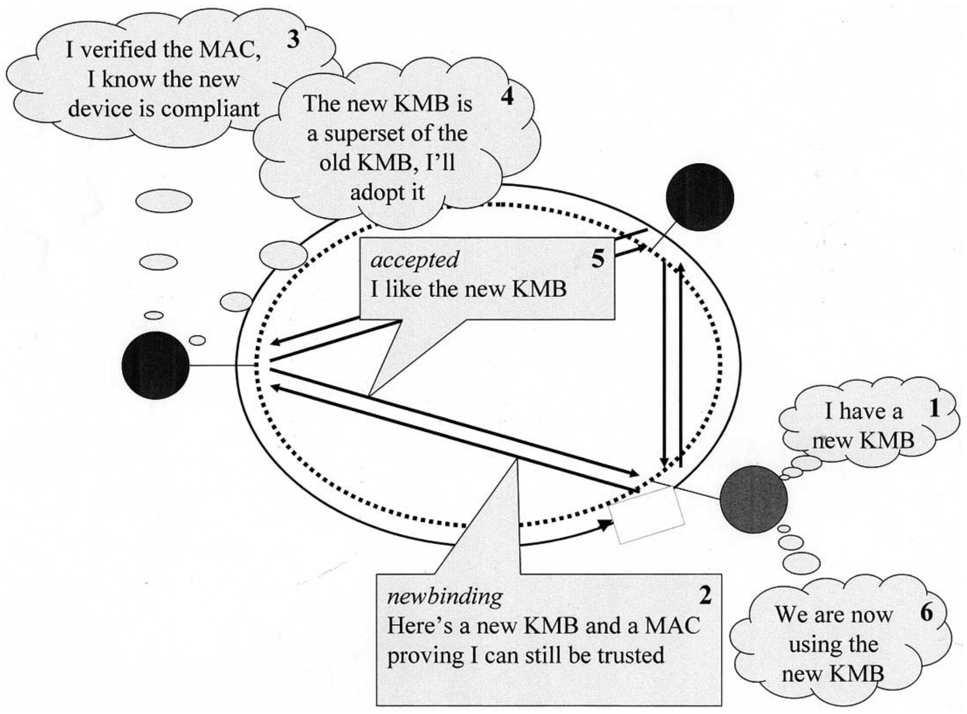
I verified the MAC, I know the new device is compliant 3
The new KMB is a superset of the old KMB, I'll adopt it 4
accepted 5 I like the new KMB
I have a new KMB 1
newbinding 2 Here's a new KMB and a MAC proving I can still be trusted
We are now using the new KMB 6

Fig. 6. New binding.

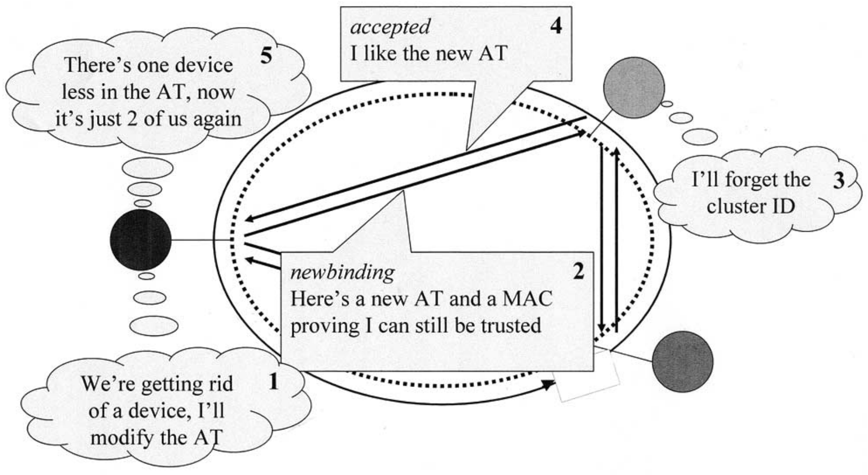
accepted 4 I like the new AT
There's one device less in the AT, now it's just 2 of us again 5
I'll forget the cluster ID 3
newbinding 2 Here's a new AT and a MAC proving I can still be trusted
We're getting rid of a device, I'll modify the AT 1

Fig. 7. Device removal.

the old cluster-wide key management block with the one the new device is proposing, so that the revocation is the sum total of both of them: a device revoked in either key management block will also be revoked in the combined key management block.[6] (See Fig. 6.)

Some servers may have access to new key management blocks from external sources; for example, a set-top box may receive key management blocks from a broadcast channel, and DVD players may see new key management blocks on prerecorded disks. These situations are treated exactly as if a new server with a new key management block had introduced into the cluster.

Consumers may want to remove devices from the home network for a variety of reasons: they may want to sell a used device, or a device may have stopped functioning or may have been lost—especially likely with portable audio players the size of a stamp. Any authorizer can remove a device from the cluster by adding a "pending removal" entry in the authorization table corresponding to the device to be removed. Because the authorization table has changed, this triggers a *newbinding* message sequence.

If the departing device is still on the network, it can accept the new binding. When the authorizer that triggered the new binding message gets this positive response, it can change the "pending removal" entry to "removed." The departing device, because it is compliant, can then "forget" the cluster ID. Moreover, a compliant device will never play content associated with an authorization table in which the device is either not listed or is listed as removed. Thus, it can no longer access content associated with the cluster. (See Fig. 7.)

[6]This is very similar to the media key block extension idea in CPRM.

On the other hand, the device may not be connected, either because it was truly incapacitated or because the consumer is trying to be sneaky. The authorizer will only allow a limited number of devices that are in this "zombie" state of pending removal.[7]

VI. Download to the Home Network

The xCP Cluster Protocol allows an anonymous trust system that is divorced from actual physical media. The DRM server can deliver and bind content to an entire home, not just a single piece of media. From the consumer's point of view, this gives him the maximum flexibility: he can play the content on any device in his (extended) house. He can also make unlimited backup copies. In fact, the content protection technology should be transparent to his normal day-to-day enjoyment of the content. It would only intervene and become an obstacle when he did something questionable, like trying to distribute it to his million closest friends on the Internet.

Technically, the xCP Cluster Protocol supports the DRM download function by having the DRM server actually join the cluster. That way the server learns the cluster ID, and thereby can calculate the cluster's binding key. As a side effect, if the server discovers that the key management block in use in the cluster is out of date, it can propose a new one. As long as the cluster does not consist solely of circumvention devices, the compliant devices will accept this "new binding," and the cluster will thereby be inoculated against new circumvention devices. If there were any existing circumvention devices in the cluster, they will not be able to convincingly fabricate an "accept" message to the "new binding." The server, of course, would bind the content to the new binding, so the existing circumvention devices would be left in the lurch.

The key management block update mechanism is not operating for the benefit of the consumers; it is operating for the benefit of the content owners. The consumers, especially those motivated by narrow self-interest, might want circumvention devices in their cluster, especially if they want to give away copies of their content to their friends. However, the *quid pro quo* is that if they want new content, they must accept that their cluster's key management block gets updated.

It is important that the DRM server, even though it is joining the cluster, not be counted as one of the devices in the home. If it counted, the cluster might prematurely reach the "N" limit requiring external authorization, an inconvenience to the consumer. No one wants to penalize consumers for buying a lot of content. The simple technical solution is for the server to identify itself as such in its "authorize me" message. Any device so identified will be automatically allowed to join the cluster, regardless of the current number of devices already in the cluster.

On the surface, this seems to open up an attack. A normal consuming device could lie about its status and claim to be a DRM server in order to allow arbitrarily large clusters (a whole university campus, for example). Understanding why and how this attack is prevented is the key to understanding how content protection really works. Remember our mantra: it is not about cryptography; it is about the license.

First, the "authorize me" message can only be fabricated by a compliant device, because only a compliant device can process the key management block and put the correct MAC on the message. But what does this compliant device "comply" with? It complies with the license that its manufacturer signed when it got a set of keys for the device to process key management blocks. That license requires that the device's "authorize me" messages cannot lie. If a manufacturer breaches his license, he is open to civil litigation from the content owners, who are so-called third-party beneficiaries in the license. Alternatively or in parallel, the lying devices might be declared circumvention devices, and excluded in future key management blocks.

It is worthwhile contrasting this with how a public-key-based system would solve the same problem. Many people have the wrong intuition that a public-key approach offers a stronger guarantee. In many cases, public-key systems use certificates not only to certify identity, but also to convey a certain level of entitlement. For example, in a public-key system, a trusted third-party authority would certify that a given certificate belonged to a DRM server provider, as opposed to a device manufacturer. Of course, the subject company could have lied to get the certificate and could have manufactured normal devices using the server certificate. As a matter of civil law, they undoubtedly violated the license they signed to obtain the certificate. As a matter of technology, their certificate is liable to be revoked in a certificate revocation list. These enforcement mechanisms are exactly analogous to the ones used in a broadcast-encryption-based system. And in both cases, the schemes are only as strong as their enforcement mechanisms.

The public-key-based system, however, is built on individual identity; the broadcast-encryption-based system is not. This does not necessarily mean the end user's identity—even a device's public-key certificate that is issued for a unique device ID raises privacy issues. Since this certificate participates in the handshake flow, there will always be a server system that can correlate all these transactions and build a profile. It is only a small step to link this information to the end user that owns the device—which could easily be performed during warranty registration or initial payment for the device. Note that with broadcast encryption, this is different: the content publisher has the guarantee that only compliant devices will get the content but, because of the one-way nature of such a system, he has no way of knowing which compliant device it will be.

We have suggested earlier that this difference offers important privacy advantages for the consumer. It is now time to explore this potential. In the anonymous trust system, the cryptographic binding step does not need to identify the user. The media ID binding is clearly anonymous. The network binding ID could be just as anonymous, although that certainly depends on how the external authorization for exceptionally large clusters is handled. But there are at least two

[7]The exception mechanism mentioned in a previous footnote can be used to override this restriction as well.

other possible points of identification when the user purchases content for electronic delivery. First, he might have to identify himself to pay, for example, with his credit card. Second, the TCP/IP address of his Internet portal to his home network might likewise identify him. We will take each point in turn.

Techniques to achieve anonymity in payment, so-called digital cash or electronic cash technologies, are well known and complete [11], [12]. It is safe to say, however, that the beta test deployments of these technologies have not yet met with commercial success. Nonetheless, these technologies offer the ultimate high-privacy solution for an anonymous trust system.

There is an alternative, however, even if the world seems fixated on identification-based payments. If content is being delivered through a subscription service, instead of a pay-for-download service, then it is possible to use broadcast encryption to divorce the payment from the actual delivery mechanism in such a way that it is not possible for the server to associate a particular piece of content with a particular subscriber. At the same time, however, the server is confident that it is delivering content only to paid subscribers.

The subscription service works as follows. The consumer pays for the subscription service in any way that seems appropriate, including credit card or monthly bill. In return, the consumer receives a set of device keys in a broadcast encryption scheme; we call it the *subscription scheme*. Content is delivered with title keys encrypted in a key management block in the subscription scheme. The DRM client would use its device keys to decrypt the title keys of the content the user wanted to download. The client would also have a set of xCP Cluster Protocol device keys to reencrypt the title keys in the binding key of the user's particular cluster. The user could use his client code to download and bind any content that his subscription entitled him to, completely anonymously. There would be no way for the server to associate download requests with any particular subscriber. On the other hand, the content is encrypted in a way that only valid subscribers in good standing can decrypt, so the server's ignorance cannot be a source of unauthorized distribution of content.

We think of this high-privacy subscription service as a particularly useful example of an anonymous trust system. However, it does complicate the client software: unlike most anonymous trust systems, in this system the client code must have and protect secret keys (the device keys). This added complexity in turn presents an opportunity: since the client software must be tamper resistant, it is possible to use it to enforce more complicated rules in the subscription service. For example, a monthly fee may entitle the subscriber to only five downloads a month, instead of unlimited downloads. Many people would suggest that anonymity would be impossible under such a circumstance, and yet the combination of tamper-resistant client software backed up by broadcast encryption's ability to revoke compromised clients makes an anonymous system of this type feasible.

What about the user identifying himself by his TCP/IP address during the download? This situation is very similar to electronic cash: technology for provably anonymous networking is well known and well developed [10], but the marketplace has been lukewarm at best to this idea. It is likely that many consumers feel an adequate degree of anonymity based on how they connect to the Internet. By the time a request reaches a server, the server usually can determine little more than what domain the request came from, i.e., what Internet service provider or company the user is belongs to. Although the service provider may be able to trace back a particular request to a unique individual, most service providers are interested in protecting the privacy of their clients, as long as they are restricting themselves to lawful activities.

Although, in the current state of the Internet, connection anonymity may be adequate for the privacy concerns of most users, we recognize that certain circumstances demand more. Two of the authors worked on a system delivering scientific papers in life sciences to subscribers in universities and pharmaceutical firms. All parties were very interested in the privacy of the end users, and that it not be possible to trace individual papers to individual recipients. The pharmaceutical companies insisted, in addition, on the privacy of their entire domain. The list of papers that an individual company was requesting would imply which diseases the company was developing drugs for, which was among the most sensitive of the company's confidential information.

VII. Conclusion

Many DRM systems make use of public-key cryptography to enforce compliance among participating devices. However, this approach has several drawbacks: the underlying calculations are computationally demanding, a bidirectional connection is required to perform a handshake protocol, and the end user's privacy can be compromised easily.

A new generation of rights management solutions leverages broadcast encryption technology to provide cryptographic enforcement of device compliance. These systems do not require a handshake and can, therefore, be used for media or broadcast-based distribution in an otherwise disconnected environment. Since they are based on symmetric key cryptography, they are better suited for integration in low-cost consumer devices. Thanks to its inherently one-way nature, broadcast encryption also provides a much higher level of consumer privacy. One drawback of broadcast encryption has always been the large amount of data necessary to represent revocation information. However, with recent advances, it is now possible to compact revocation data to the point where storage requirements are similar to that of a public-key certificate revocation list.

Several broadcast-encryption-based rights management systems are currently in use for distribution of content in consumer products like DVD-audio players or DVD recorders. However, we believe that this technology has high potential for nonmedia-based forms of distribution as well. Broadcast encryption is the basis of the xCP Cluster Protocol, which uses this technology in an innovative way to bind content to the home network. Broadcast encryption can also be used to build a flexible distribution system that allows download of content to recordable media like DVD-RAMs or to an xCP home network. As more and more of these solutions are built on the same key management infrastructure, revocation information propagates more rapidly through the system and provides for even more effective exclusion of compromised devices. All of these broadcast-encryption-based technologies provide a high level of privacy protection and a guarantee of anonymity as long as a payment process does not require the user to provide identification.

We conclude that broadcast-encryption-based rights management systems are superior to solutions that are based on public-key cryptography, since the former are more efficient, support disconnected distribution, and provide a higher level of consumer privacy. These advantages of broadcast encryption not only apply to the currently available media-based distribution systems but also to other areas of DRM.

References

[1] A. Fiat and M. Naor, "Broadcast encryption," in *Lecture Notes in Computer Science, Advances in Cryptology*. Heidelberg, Germany: Springer-Verlag, 1994, vol. 773, pp. 480–491.

[2] Key management for multicast: Issues and architectures, D. M. Wallner, E. J. Harder, and R. C. Agee. (1997). [Online]. Available: ftp://ftp.ietf.org/internet-drafts/draft-wallner-key-arch-01.txt

[3] C. K. Wong, M. Gouda, and S. Lam, "Secure group communications using key graphs," in *Proc. SIGCOMM*, 1998, pp. 68–79.

[4] D. Naor, M. Naor, and J. Lotspiech, "Revocation and tracing routines for stateless receivers," in *Lecture Notes in Computer Science, Advances in Cryptology*. Heidelberg, Germany: Springer-Verlag, 2001, vol. 2139.

[5] B. Traw, "Protecting digital content within the home," *IEEE Computer*, vol. 34, pp. 42–47, Oct. 2001.

[6] Microsoft's digital rights management scheme—Technical details [Online]. Available: http://cryptome.org/ms-drm.htm

[7] Content protection for recordable media: Introduction and cryptographic elements [Online]. Available: http://4centity.com

[8] Content protection for prerecorded media: Introduction and cryptographic elements [Online]. Available: http://4centity.com

[9] J. Yoshida. (2002, Jan.) Content protection plan targets wireless home networks. *EETimes* [Online]. Available: http://www.eetimes.com/story/OEG20020111S0060

[10] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 4, no. 2, pp. 84–88, Feb. 1982.

[11] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *Lecture Notes in Computer Science, Advances in Cryptology*, S. Goldwasser, Ed. Heidelberg, Germany: Springer-Verlag, pp. 319–327.

[12] ECash.com [Online]. Available: http://www.ecash.com

[13] J. Lotspiech, S. Nusser, and F. Pestoni, "Broadcast encryption's bright future," *IEEE Computer*, vol. 35, pp. 57–63, Aug. 2002.

[14] G. Itkis, "Protecting recordable media," presented at the Meeting Content Protection Working Group, Burbank, CA, 1999.

Jeffrey Lotspiech (Member, IEEE) received the M.S. degree from the Massachusetts Institute of Technology, Cambridge, in 1972.

He is currently a Research Staff Member at IBM Almaden Research Center, Henderson, NV. His research interests include content protection and software tamper resistance.

Stefan Nusser received the Ph.D. degree in management information systems from Vienna University of Business Administration and Economics, Vienna, Austria, in 1997.

He is currently a Research Staff Member at IBM Almaden Research Center, San Jose, CA. His research interests include enterprise content management, content protection, and digital rights management.

Florian Pestoni received the M.S.E.E. degree from the University of Buenos Aires, Buenos Aires, Argentina, in 1991 and the M.B.A. from the University of California, Berkeley, in 2001.

He is currently a Software Engineer at IBM Almaden Research Center, San Jose, CA. His research interests include digital content distribution.

Mr. Pestoni is on the Program Committee for the ACM Workshop on Digital Rights Management.