A DRM Security Architecture for Home Networks

Bogdan C. Popescu
Bruno Crispo
Andrew S. Tanenbaum
Vrije Universiteit Amsterdam
The Netherlands
[bopescu, cripo, ast]@cs.vu.nl

Frank L.A.J. Kamperman
Philips Research
Eindhoven
The Netherlands
frank.kamperman@philips.com

ABSTRACT

This paper describes a security architecture allowing digital rights management in home networks consisting of consumer electronic devices. The idea is to allow devices to establish dynamic groups, so called "Authorized Domains", where legally acquired copyrighted content can seamlessly move from device to device. This greatly improves the end-user experience, preserves "fair use" expectations, and enables the development of new business models by content providers. Key to our design is a hybrid compliance checking and group establishment protocol, based on pre-distributed symmetric keys, with minimal reliance on public key cryptographic operations. Our architecture does not require continuous network connectivity between devices, and allows for efficient and flexible key updating and revocation.

Categories and Subject Descriptors

J.7 [**Computer Applications**]: Computers in other systems—*Consumer products*

General Terms

Security, Design, Management

Keywords

DRM Architectures, Digital Content Protection, Compliant CE Devices

1. INTRODUCTION

In the past years there has been an increasing interest in developing digital rights management (DRM) systems [20, 10, 11]. The main purpose of a DRM system is providing digital data content (mostly home entertainment-related) in a way that protects the copyrights of content providers (CPs) and to enable options for new business models for content distribution and access.

DRM'04, October 25, 2004, Washington, DC, USA.

Consumers want to enjoy content without hassle and with as few limitations as possible. They want to network their devices and easily access any type of content in their home environment. Experience has shown strong negative consumer reaction when copyright protection mechanisms have disrupted interoperability expectations [14]. The content industry, however, wants to protect its digital assets. One solution addressing both these two requirements is to organize compliant devices into home content delivery networks [21] where legally acquired digital content can freely be played by any device part of the network.

DRM systems make providers less reluctant to publish content electronically and, in the end, give consumers a more versatile offering of content. DRM systems, however, are not on the consumers feature list when buying new devices; consumers simply do not have the motivation to spend extra money for DRM-enabling functionality. Besides this, in the consumer electronics (CE) business even marginal cost reductions can lead to competitive advantage. Therefore, besides being secure, implementation of DRM functionality has to be cost efficient. In this context, security mechanisms that rely on public key cryptographic operations are seen as a disadvantage, since they normally require (more expensive) cryptographic accelerator hardware in order to operate reasonably efficient.

This paper describes a security architecture for the "Authorized Domains" framework [9] introduced by the DVB consortium [1] as means to facilitate the creation of home content delivery networks of CE devices. The foundation of our design effort is a novel compliance checking protocol which allows compliant devices part of a domain to individually authenticate each other **without** relying on expensive public key cryptographic operations. An additional benefit of our protocol is that it supports **efficient** and **flexible** revocation of compromised devices.

The rest of the paper is organized as follows: in Section 2, we elaborate more on the issue of compliance checking in DRM architectures. In Section 3, we give a quick overview of the "Authorized Domains" framework. In Section 4 we describe the new security architecture, showing how we deal with issues such as domain creation, new device registration, secure content storage, key update and device revocation. Finally, in Section 5 we talk about the performance implications of our design, in Section 6 we review related work, and in Section 7 we give our conclusions.

2. DRM AND COMPLIANCE CHECKING

DRM systems rely on the fact that they operate on so-called "compliant devices". The most important property of such devices is the fact they are *self-policing* - before performing any operation on a piece of data content, they check that the operation does not contradict the rules set by the content owners for that piece of content. For example, a compliant video recorder will never make a copy of a piece of video marked "no copy", although it has the ability to do it.

Currently, there are two possible approaches for doing device compliance checking: in the case of *individual authentication*, this is done by means of public key cryptography - by assigning each device a unique public/private key pair with the public key certified by a licensing organization through a digital certificate. In this case, whenever two compliant devices need to interact, they must first engage in a mutual authentication protocol, proving to each other they have the private keys corresponding to "compliant" public keys.

The other way to do device compliance checking is through *group authentication*: in this case, the identity of a given device is un-important, as long as the device can prove it is part of the group of compliant devices. In practice, the most efficient way to do group authentication is based a class of symmetric key encryption algorithms known as *broadcast encryption* [17].

The basic idea behind broadcast encryption [12, 25, 19] is to allow a dynamic group of entities (compliant devices in the DRM scenario) to establish a common secret, by receiving messages broadcast by a group controller (the licensing organization in this case). Once a common group key has been established, it can be used to protect the digital content exchanged by the compliant devices part of the group.

In a broadcast encryption algorithm specifically designed for DRM applications [19], key material is organized in a logical binary tree, where each node in the tree corresponds to a symmetric key. The number of leaves in the tree is equal to the maximum number of compliant devices in the world; this may be in the order of hundreds of million of even more in the case of very successful products. Each device is assigned a leaf, and contains all the (secret) keys that are on the path between its assigned leaf and the root of the tree (thus, the root key is known to all devices). At the beginning, all devices are part of the group; the group key is encrypted under the root key. Once circumvented devices are identified, they are revoked (excluded from the group) by the licensing organization, which generates a new group key and encrypts it with keys in the tree that cover only leaves corresponding to correct devices; circumvented devices cannot recover the new group key. This scheme works quite well when there are few revoked devices: in this case, a small number of sub-trees suffice for a complete group cover. However, as more devices are revoked, more and more small sub-trees are needed to cover only "good" leaves, so the group key needs to be encrypted with many keys, which leads to a large broadcast message size. To handle this issue, a number of variations of the basic broadcast encryption scheme have been proposed [19], based on alternative tree covering algorithms; in this way, even large number number of devices can be revoked without requiring an un-acceptably large broadcast message size.

Our security architecture relies on a compliance checking mechanism based on individual device authentication. Although this requires public keys, through careful design it is possible, as we will show in Section 4, to minimize the impact public key cryptographic operations have on system performance, up to the point where use of hardware cryptographic accelerators can be avoided. Before getting into the design details, we will first introduce our system and trust model and discuss the possible attack scenarios.

3. THE "AUTHORIZED DOMAINS" FRAMEWORK

The "Authorized Domains" framework [9] has been first introduced by the DVB consortium [1] as a means to facilitate the creation of secure home networks of consumer electronic devices. Compliant devices owned by one household connect together to form one authorization domain. Legally acquired digital content can then seamlessly flow from device to device inside the domain, but tight controls are applied at the domain borders, in order to prevent illegal content distribution. As said previously, this balances the interests of content owners (who want protection of their copyrights) and content consumers (who want unrestricted use of the content they paid for).

3.1 Design Requirements

When designing the security architecture for the "Authorized Domains" framework, we had to consider a number constraints dictated by the deployment environment (home networks) as well as by the need to limit the manufacturing costs. More specifically:

- Continuous network connectivity among all devices in one domain cannot be assumed; for some devices, operating in disconnected mode is the norm, rather than the exception: this is the case of PDAs, personal music players, car stereos, etc.
- The existence of secure clocks **cannot** be assumed. Adding tamper-resistant hardware clocks to consumer electronic devices would un-desirably increase the overall price of these devices.
- Given the cost constraints, devices should not require cryptographic hardware accelerators in order to operate efficiently.
- Based on previous experience with pirated set-top boxes for pay-per-view TV, it can be expected that counterfeit devices will become available at some moment following the introduction of compliant devices on the consumer market. It is therefore essential that our design allows revocation of potentially large numbers of counterfeit devices without significant performance degradation.

Because we cannot rely on hardware accelerators, the only option for performing cryptographic operations is the general-purpose CPU embedded in the device. Existing consumer electronics products (DVD players, TV's, handhelds, etc.) typically use embedded (16/32 bit) RISC processors for general-purpose operations; dedicated tasks (e.g. video processing) is typically done on dedicated hardware. The clock speed of these general-purpose CPUs ranges from tens to hundreds of MHz. An RSA (1024 bit) sign operation may

therefore take between tens of milliseconds to several seconds (full load), depending on the platform. Taking into account that general-purpose CPUs are primary used for non-cryptographic purposes (mostly control), and under normal device operation they are loaded to a large extent, and given the fact that RSA 1024 bit keys may not be suitable for a 10 years lifespan, it is safe to assume that, at least for mid- to low-end devices, a public key authentication protocol may take in the order of seconds to complete. From a consumer point of view, latency in the order of seconds during normal operation of CE devices is not acceptable; therefore, one of our prime design goals is to minimize the number of public key operations a device needs to perform.

On the other hand, the lack of continuous network connectivity among all devices in the domain makes impractical to use some of the centralized content distribution/storage mechanisms employed by other DRM architectures [10]. For instance, storing content in a central repository, encrypted under a key shared by all devices in the domain is not an option. Our design needs to focus on mechanisms that allow content to seamlessly "float" from device to device. For example, a user should be able to download her songs from her personal music player to her car stereo, even when both these devices are disconnected from the rest of the authorized domain.

3.2 System Model

For the security architecture we describe in this paper, we consider a high-level system model consisting of the following entities:

- A number of **content providers**. These are organizations/companies interested in selling **digital content items** (usually home-entertainment related) to **consumers**. Content providers associate **usage rules** with the content they deliver; associating usage rules with content helps enforcing providers' copyrights, and facilitates a variety of business models (e.g. pay per view, subscription, etc.).

- A number of **CE manufacturers** that produce and sell **compliant devices**. These devices render digital content for consumers, while enforcing the usage rules set by content providers.

- A **licensing organization** that certifies compliant devices and revokes the circumvented ones. Usually, the licensing organization delegates the certification task to licensed CE manufacturers that are contractually bound to only produce compliant devices according the specified robustness rules.

- A number of **authorized domains** (ADs). Each AD consists of compliant devices owned by one household, and forms the authorization unit of the system, in the sense that usage rules associated with content apply to one AD as a whole.

Figure 1 shows the internal structure of an AD. As we can see, it consists of a number of compliant devices whose main purpose is to render digital content, which seamlessly moves from device to device inside the domain. In addition to rendering content, a device may play the **AD Manager** role, or a **Content Manager** role, which are as follows:

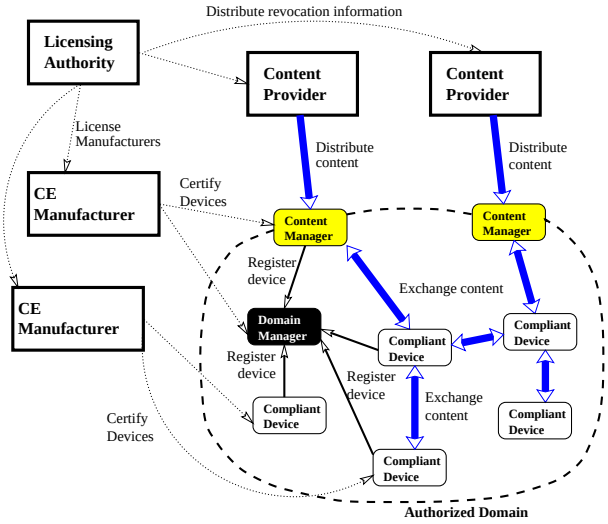

Figure 1: AD structure and interaction with external entities

- The AD Manager keeps track of the other devices in the domain: it registers new devices, and removes the ones leaving the domain (either voluntarily, or because they have been revoked by the licensing organization). There can be only one manager per AD. If multiple devices in the domain have this capability, the user must select one of them as the active one.

- A Content Manager brings new data content into the domain by interacting with content providers. Different providers may choose different types of devices to supply their content; as a result, there can be multiple content managers inside a domain.

It is important to understand that a device can play multiple roles: it can render content, as well as being the AD manager and possibly a Content Manager. The amount of functionality packed in a given device is a manufacturer/consumer choice. From the consumer point of view, extra functionality in a device is materialized through additional command interfaces: the AD manager device needs a special AD management interface, while the content managers need command interfaces allowing interaction with the content providers they support.

At manufacture time, each compliant device is given a public/private key pair, with the private key stored in tamper-resistant memory, and the public key certified by the manufacturer by means of a device certificate. Each compliant device is identified by a unique *global device Id* (**GDI**), also included in the device certificate. The GDI consists of two parts - the *manufacturer prefix* - a short number identifying the CE manufacturer that produced the device, and the *device serial number* which uniquely identifies the device for that manufacturer.

Finally, we want to stress that our system model only considers on-line digital content distribution; we do not consider content that comes on pre-packaged media (e.g. CDs and DVDs), which, at least under current distribution models, is not very well suited to support fine-grained DRM.

3.3 Attack Scenario and Trust Model

The attack scenario we consider in this paper is realistic with respect to digital content distribution: a malicious user is attempting to gain access to content to which she is not entitled. To accomplish this goal, the attacker has full control of the intra-domain home network, and can make use of compromised and circumvention devices (devices mimicking compliant devices). However, we assume the attacker has limited computational resources (cannot break cryptography), and has only limited capability of disrupting external network communication (between entities outside its home). Besides the attacker, our system includes a number of other entities: the Licensing Organization, Content Providers, CE Manufacturers, as well as compliant devices (possibly with extended functionality, such as AD Manager or Content Manager). We will now describe the trust relationships between these entities, and how they collaborate to prevent the attack scenario we introduced.

Central to our trust model is the Licensing Organization, identified through its public key. This public key is the root of trust in the system and is assumed to be known by all other parties. The licensing organization has two main functions: certifying CE manufacturers and issuing fresh device/manufacturer revocation information. Certifying CE manufacturers involves issuing a digital certificate binding the manufacturer prefix to the manufacturer's public key. The manufacturer can then use this certified key to issue device certificates. The revocation information consists of a *Global Device Revocation List* (**GDRL**); this list contains the GDIs of devices known to be no longer compliant. The mechanisms for identifying compromised devices are beyond the scope of this paper, but they would most likely involve forensic examination of illegal devices sold on the black market (illegal devices incorporating cryptographic material extracted from compromised compliant devices).

Content Providers are only interested in the correct delivery of the content they own, so for this reason they do not have to trust each other. Correct delivery means the content is only received by compliant devices which are trusted to enforce the usage rules. Content Providers deliver their content to Content Manager compliant devices over secure communication channels (authenticated and confidential). Content Providers also periodically receive fresh revocation information from the Licensing Organization (this is assumed to happen over a secure and reliable communication channel, so we do not have to worry about DoS attacks). Providers then use this information to determine whether the content manager devices they interact with are still compliant, and stop delivering content to compromised managers. The same revocation information is also bundled with the digital content supplied to content managers; this ensures that the only way to obtain new content also implies the delivery of a fresh revocation list. In this way, the communication channel between the provider and the content manager needs not to be reliable: a DoS attack aiming at preventing the device from receiving the revocation list would render the device useless, since it would not be capable of receiving new content.

Compliant devices are fully trusted as long as they are authenticated and not revoked. Domain manager devices are trusted to correctly authenticate devices before accepting them in the domain, as well as to keep up to date with the revocation information received through content manager devices, and to promptly exclude from the domain any revoked devices already part of it. On the other hand, it is possible that the domain manager itself is compromised. To counter this threat, content manager devices are trusted to correctly report the identity of the manager of the domain they are part of to content providers; a provider will then stop delivering content to a domain managed by a compromised device.

4. PROPOSED SECURITY ARCHITECTURE

In this section we describe a security architecture to fit the "Authorized Domains" framework. The key idea is to use a hybrid public key/symmetric key compliance checking protocol that greatly reduces the frequency of public key cryptographic operations needed for intra-domain device authentication.

4.1 Authorized Domain Creation

Creating a new AD requires one compliant device with AD manager functionality. When creating the new domain, the AD manager device first erases all information about the previous AD it has managed (if any); following that, it generates a *master device key* list (a list of 128 bit AES keys) which is stored in its tamper-resistant memory. The size of this list is equal to the maximum number of devices allowed in the domain; this is a manufacturer/content provider choice, but we expect it to be in order of tens. Finally, the manager generates a *domain ID*, also stored in its tamper-resistant memory. The domain ID is built as a concatenation of the manager's GDI and an ever-increasing *domain version number*. At manufacture time, the domain version number is set to zero; whenever the AD manager is reset, the domain version number is incremented, which ensures the manager will always generate different domain IDs.

Once both the master device key list and the domain ID have been generated, the AD creation process is complete, and the manager can populate the new domain by registering new devices.

4.2 Device Registration

A device that enters the AD needs to be registered with the AD manager. The registration phase consists of two steps: *compliance checking*, and *authorization*. The complete registration protocol between the AD manager (M) and a device A is the following:

Notation

$cert_E$	entity $E's$ public key certificate.
Y_E/x_E	entity $E's$ public/private key pair.
N_E	a random nonce generated by entity E.
$\{data\}_K$	*data* encrypted with the symmetric/asymmetric key K; public key signing is represented as encryption with a private key.
$[data]_K$	*data* transmitted over a secure channel protected (integrity&confidentiality) by a symmetric key K.

Protocol

(1) A $\longrightarrow$ M: $cert_A, \{N_A, GDI_M\}_{x_A}$
(2) M $\longrightarrow$ A: $cert_M, \{N_M, GDI_A, N_A, \{k_s\}_{Y_A}\}_{x_M}$
(3) A $\longrightarrow$ M: $\{N_M, GDI_M\}_{x_A}$
(4) M $\longrightarrow$ A: $[LDI_A, K_A, credentialsSet_A]_{k_s}$

4.2.1 Compliance Checking

Compliance checking is done in steps (1) to (3) of the registration protocol, and is based on the X.509 strong authentication protocol [18]. The two devices exchange their device certificates and authenticate each other. Because we cannot use secure clocks, the authentication protocol is based on random nonces. At the end of step (3), each device is assured the other party has access to a private key corresponding to a public key certified as compliant by the licensing organization. The two parties also agree on a symmetric session key k_s (a 128 bit AES key) which is used by the manager to protect the authorization information sent to the device in step (4).

After completing step (3) of the protocol, the AD manager selects the next un-used key in its master key list. This key becomes *A's master key*, and the index of this key in *M's* master key list becomes *A's local device Id* (**LDI**) in the domain (these are denoted as K_A and LDI_A in step (4) of the protocol). At this point M also needs to update its internal records, to keep track that LDI_A has been associated with GDI_A.

The first three steps in the registration protocol require public key cryptographic operations. In most cases, these operations need to be performed in software, on general-purpose CPUs, since it cannot be assumed that all devices are equipped with hardware cryptographic accelerators. As a result, registration is likely to be a slow procedure; however, since this is a relatively rare event (it happens only when the user buys a new device), the delay introduced should be acceptable.

4.2.2 Device Authorization

Accepting a device in an AD implies authorizing the device to interact with other devices in the AD in order to obtain content items. In the authorization step, the manager issues the new device an *authentication credentials set*, which is sent to the new device in step (4) of the registration protocol, together with its LDI and device master key. The authentication credentials set consists of a number of (*authentication key*, *authentication ticket*) pairs.

Authentication keys are symmetric keys shared between two devices part of the same AD. Each device is given authentication keys for every other device already part of the domain **as well as** for all **potential** devices that may join the AD in the future (thus, the number of authentication keys given to each device is equal to the size of the master key list generated by the manager when creating the AD). In this way, when new devices join the AD, existing devices need not be updated, which allows the AD to operate even without assuming continuous network connectivity among all devices.

There is an authentication ticket associated with each authentication key. The *(authentication key, authentication ticket)* pair allowing device A to authenticate to a device B has the form $(K_{AB}, \{K_{AB}, ID_{Domain}, GDI_A, LDI_A, LDI_B\}_{K_B})$, where K_{AB} is a 128 bit AES key, and K_B is the master device key for B. The authentication can be used by A to prove to B that it is a compliant device part of the same domain. Since the ticket is encrypted with a key shared only between B and the manager, B is assured only the manager could have created it, which in turn (given the manager is a compliant device following the protocol) implies the manager has verified the compliance of A.

Once a device is part of the domain, it can be used to process the content items it acquires from other devices in the domain. Before exchanging content items, two devices authenticate each other in order to prove they are part of the same domain. The authentication protocol between two compliant devices part of the same AD is shown in Table 1. The protocol has been first introduced in [8] and is based on a variation [5] of the classical Kerberos authentication protocol [15]. It relies on the security property of *keyed hash functions* used as a basic primitive to generate fresh session keys, and works as follows:

(1)	A	$\longrightarrow$ B:	LDI_A, N_A
(2)	B	$\longrightarrow$ A:	$LDI_B, N_B, authenticationTicket_{BA}$
(3)	A	$\longrightarrow$ B:	$\{N_B\}_K, authenticationTicket_{AB}$
(4)	B	$\longrightarrow$ A:	$\{N_A\}_K$

Table 1: Device-to-device authentication protocol

In the above protocol, K = *SHA-1*$(K_{AB}, K_{BA}, N_A, N_B)$, where *SHA-1* is the secure hash function described in [2]. We assume that initially A and B are complete strangers (they do not know each other's LDIs). At the end of the protocol K is the shared secret between A and B and can be used for securing the data traffic between the two devices.

During the authentication protocol, before accepting the other party's ticket, a device needs to do the following checks:

- The ID_{Domain} in the ticket corresponds to the authorized domain the device is part of.
- The second LDI value in the ticket is equal to its own LDI.
- The *SHA-1* hash of the device description sent by the other device matches the hash in the ticket.
- The other device has not been revoked (we will show later how revocation checks are performed)

4.3 Device Removal

There are three cases a device is removed from a domain: when the device is moved to another domain (voluntary leave), when the device is no longer functional (damaged/stolen devices), and finally when the device is known to be no longer compliant (device revocation).

4.3.1 Voluntary Leave

In this case we assume a connection between the device and the domain manager. The two devices authenticate each other, and following that, the the manager obtains the GDI of the departing device. This GDI is then added to the domain's local revocation list which will be described later.

4.3.2 Damaged/Stolen Devices

In this case, we cannot assume a connection between the device and the domain manager. In order to remove a device, the domain manager should provide a user interface allowing the domain owner to identify the device to be removed (this can be a display showing the list of all devices in the domain, with some input mechanism that allows the owner to select from the list). Once the domain owner has identified the device to be removed, the manager adds that device's GDI to the local revocation list.

4.3.3 Device Revocation

Devices known to be no longer compliant are revoked by the licensing organization by having their GDIs listed on the global device revocation list (GDRL). Since it cannot be assumed all compliant devices incorporate secure clocks, device revocation lists are distributed by content providers together with the data content items; thus there are no "freshness" requirements regarding revocation information, except that the only way to obtain new content automatically updates the GDRL.

Revocation lists can grow very large, since they contain information regarding **all** compromised devices in the world (if we have one billion compliant devices, out of which only 1% are compromised, the size of the revocation list would be in the order of 40MB). Because of this, we cannot assume that all devices have enough memory/computational power to process the global revocation list.

4.4 Revocation Mechanisms

As discussed in Section 3.3, it is content manager devices that bring fresh revocation information in the domain (this revocation information is bundled with the digital content supplied by content providers). Content managers also report the identity of the domain manager to the provider, which, before supplying any new content, ensure that the AD manager has not been revoked. Once a content manager device receives a new GDRL, it does the following:

- The content manager attempts to connect to the AD manager.
- If the AD manager is reachable, the content manager forwards it the GDRL; the AD manager processes the GDRL, and returns a *Local Revocation List* (LRL), which is then bundled with the data content (in this case, the content is dubbed **lightweight**).
- If the AD manager is not reachable, the content manager keeps the original GDRL attached to the data content (in this case the content is dubbed **heavyweight**).

It is important to understand that a LRL is only meaningful for devices part of the domain whose manager has issued that LRL. Should a piece of data content have to be exported to other domains, it should be the GDRL and not the LRL that is attached to that content.

4.4.1 The Local Revocation List

The AD manager is responsible with generating the local revocation list. This list consists of the GDIs of domain devices that have been either revoked (they are present in the GDRL) or have been removed from the domain. Since the total number of devices in a domain is at most in the order of hundreds, and adding/removing devices from a domain are rare events, we expect the LRL to be much smaller than the GDRL.

It should be possible for every device in the domain to authenticate a LRL as produced by the AD manager. To accomplish this, the AD manager creates one *LRL authentication code* for each device (and potential device) in the domain. For a device with LDI I the LRL authentication code is the HMAC-SHA-1 [16] of the LRL using the master key K_I. The LRL then consists of the actual list of revoked devices plus the authentication codes for all keys in the master key list. A device can check the authenticity of the LRL by first finding the LRL authentication code corresponding to the device's LDI, and then verifying that the authentication code is identical to the HMAC of the list (HMAC computed using its own device master key).

4.4.2 Restricting Content Distribution

It is important that revoked devices cannot receive new digital content, so they eventually become useless. In order to ensure this, a compliant device is allowed to re-distribute content to other devices in the domain only if it is capable of interpreting the revocation information bundled with the content. In the case of lightweight content, this is always the case; for heavyweight content, we expect that only a limited number of powerful devices will be able to process the GDRL. However, even if a device is not capable to process the GDRL attached to a content item, it can still render the item; the only limitation is that it cannot further distribute the content to other devices in the domain.

Compliant devices may attempt to convert heavyweight content to lightweight by contacting the AD manager in order to obtain the LRL for that content item. Once the conversion succeeds, any device in the domain is allowed to participate in the distribution of that item.

We can now see the clear advantage of our revocation scheme: with traditional revocation mechanisms (global revocation lists, or the broadcast encryption schemes), the amount of information that needs to be transmitted and processed by **all** devices grows linearly with the **total** number of revoked devices. On the other hand, our two-level revocation list scheme only requires a small fraction of powerful devices to retrieve and process global revocation information. The majority of compliant devices only have to deal with local revocation lists, which are orders of magnitude smaller than the global ones.

4.5 Key Update

If too many devices are removed from the domain, the domain manager may eventually run out of master keys to assign to new devices. One solution to this problem is to terminate the domain and re-start with a new master device key list. However, this is not exactly user-friendly.

A more acceptable option is to re-use the LDIs of removed devices. Consider a device A, with $LDI_A = I$. When A is removed from the domain, its GDI is added to the domain's LRL, and $A's$ device master key is replaced with a fresh key in the manager's master key list; this new key is then assigned to the next device joining the domain (assume this is B). In this way, B is now assigned the LDI previously assigned to A ($LDI_B = I$). This does not interfere with device revocation, since it is the GDI, and not the LDI that is added to the LRL. As in the normal device registration protocol, the manager gives B an authentication credentials set for all the other master keys in its master key list. The problem now is that all the other devices in the domain have tickets encrypted with $A's$ old master key instead of $B's$ key, and they need to be updated. However, this update is done by B itself in an incremental manner, the first time it needs to interact with other devices already part of the domain. For this, during registration, the AD gives B the new $(authenticationKey, authenticationTicket)$ pairs for all devices already part of the domain, each pair encrypted under

the master key of the respective device. The new authentication protocol between B and another device C already part of the domain becomes then:

(1) B $\longrightarrow$ C: LDI_B, N_B
(2) C $\longrightarrow$ B: $LDI_C, N_C, authenticationTicket_{CA}$
(3) B $\longrightarrow$ C: $\{K_{CB}, authenticationTicket_{CB}\}_{K_C}$, $authenticationTicket_{BC}$
(4) C $\longrightarrow$ B: $\{N_B\}_K, authenticationTicket_{CB}$
(5) B $\longrightarrow$ C: $\{N_C\}_K$

In step (2) of the protocol, device C forwards B its old credentials (for A), since B is reusing $A's$ LDI. B attempts to decrypt and (authenticate) the ticket, but since the ticket is encrypted with $A's$ old master key, the operation fails. B recognizes that C has not been updated, and forwards it the update ($\{K_{CB}, authenticationTicket_{CB}\}_{K_C}$) which it has obtained from the domain manager (and is encrypted with $C's$ master key). In step (3), C decrypts and authenticates the update using its master key, and replaces the corresponding entry in its credentials set with the data in the update packet. Following this, C uses the (updated) key K_{CB}, together with the key K_{BC} retrieved after decrypting $authenticationTicket_{BC}$ to compute the shared key K =*SHA-1*(K_{BC}, K_{CB}, N_B, N_C), which is then used to encrypt $B's$ challenge. Finally, in step (5), B has all the information needed to compute K, which it uses to encrypt $C's$ challenge, and complete the protocol. Before accepting the other party's ticket, both B and C also need to perform the checks described in Section 4.2.2, in order to make sure the ticket has been issued for the right domain, and the device has not been revoked. In addition to these checks, the device receiving the new credentials should also check at step (3) of the protocol that the credentials it has received have not been revoked.

4.6 Secure Content Storage

Data content items are brought in the domain by the content manager devices. They bring this content by interacting with external content providers. Data items are stored in un-encrypted form only in tamper resistant memory. Given the fact that tamper-resistant memory is considerably more expensive than un-trusted storage, we employ a two level scheme: once a content manager obtains a piece of data content, it generates a random *content key* (a 128b AES key), and encrypts the content with that key. Following that, it encrypts the content key with its master key. The *(encryptedContent, encryptedContentKey)* tuple can then be safely stored on insecure storage. Whenever the device needs the content, it can read the *(encryptedContent, encryptedContentKey)* tuple in its tamper resistant memory, use its master device key to decrypt the content key, and use the content key to decrypt the actual content.

The same optimization can be used to improve the performance of content transfer between devices. Assuming two devices A and B part of the same domain, the protocol for securely transferring content from A to B is as follows:

- A and B authenticate each other as part of the same domain and establish a secure communication channel.
- A transfers the encrypted content to B over an insecure channel (this is safe since the content is encrypted with the content key).
- A decrypts the content key with its master key, and transfers the content key to B over the secure channel.
- B encrypts the content key with its master key, and stores it (together with the encrypted content) on its insecure storage for later use.

5. DISCUSSION

The great advantage of the security architecture described in this paper is that public key operations are only infrequently required. In fact, for a non-manager device, the only time it needs to perform public key operations is during the registration phase, for authentication to the domain manager. Following that, all authentication between devices part of the same domain is done by means of (fast) symmetric key operations. The price we pay for this is additional storage requirements in every device; however, assuming authorized domains only contain a limited number of devices (in the order of tens), these storage requirements are not excessive. Furthermore, devices only need tamper-resistant memory for storing their device master key. All the other data can be stored in un-trusted memory, encrypted under the master key.

Table 2 lists the memory requirements for domain devices. We assume 128 bit AES keys are used, global device identifiers are 64 bit long (this allows for more than a trillion devices), domain version numbers are 16 bit long (a device manager can create 65536 domains during its lifetime), and local device identifiers are 8 bit long (up to 256 devices per domain). For these numbers, the size of the domain Id is $64b + 16b = 80b$, while the size of an authentication ticket $authTicket_{AB} = \{K_{AB}, ID_{Domain}, GDI_A, LDI_A, LDI_B\}_{K_B}$ is $128b + 80b + 64b + 8b + 8b = 288b$. Based on this, we calculate the following storage requirements:

Max. no. devices:	N
No. of revoked devices:	R
Master key list size:	$N * 128b$
Auth. ticket size:	$288b$
Auth. cred. set size:	$N * (288b + 128b) = N * 416b$
Ticket revocation list size:	$N * 160b + R * 64b$

Table 2: Memory requirements - generic case

We instantiate these generic number for three particular cases: small domains (up to 20 devices with at most 10 removed/revoked devices), large domains (up to 100 devices with at most 50 removed/revoked devices) and large domains with frequent device removal (up to 100 devices with at most 500 removed/revoked devices). The numbers we obtain are shown in Table 3:

Max. no. devices	20	100	100
No. revoked devices	10	50	500
Master key list	320B	1600B	1600B
Auth. ticket	288b	288b	288b
Auth. cred. set	1040B	5200B	5200B
LRL size	480B	2400B	6000B

Table 3: Memory requirements - specific scenarios

One limitation of our architecture is that public key authentication is still required for device registration. How-

ever, since registration is a rare event, we believe it is acceptable from the user's point of view to have a rather slow device registration process, as long as all further device interactions (once the device is part of the domain) are lightweight and fast.

Another limitation is related to the domain size. Given the key pre-distribution scheme we employ, the size of the authentication credentials set for a given device is proportional to the maximum number of devices in the domain. Thus, given the storage constraints associated with CE devices, the maximum domain size is restricted to tens, maybe hundreds of devices. However, this should not be much of a problem, given that our protection architecture is specifically designed for home networks. For other types of DRM domains (for example, a university campus, or a large company), alternative protection architectures need to be considered.

Finally, our architecture makes use of global device identifiers, which in theory could allow content owners to track content consumption. We believe that the best option for consumer privacy protection is through an appropriate legal framework that would regulate such tracking. This is particularly important, given the fact that even without global identifiers, content consumption tracking is still possible given the architectural design of existing home content delivery networks (for example, a cable operator may be able to track pay-per-view requests by simply recording the origin of the requests). Furthermore, it is always possible to extend our basic protection architecture with privacy-enhancing features, as suggested in [7].

6. RELATED WORK

The concept of Authorized Domain (AD) has originated from work on content protection in the home environment [20, 10, 13]. Early content protection mechanisms did little to address issues such as consumer convenience and "fair use"; they could also support only a limited number of business models. Home content delivery networks were meant to fix these problems: the idea was to allow content sharing among devices owned by the same household without restrictions (or at least with as few restrictions as possible), but to carefully control content sharing between different households. The Digital Video Broadcasting (DVB) [1] standardization body later called this the "Authorized Domain" concept [9]; based on this concept, a number of home content delivery architectures have been proposed [4, 3, 24, 22].

The SmartRight system [4] has been proposed by Thompson Electronic, and relies on smart cards modules incorporated into CE devices. Their security architecture shares a number of features with ours, in the sense that basic compliance checking also relies on public key certificates issued by a licensing organization. Once devices are accepted in one domain, they all share the same symmetric domain key which is used to encrypt the protected content. Since adding/removing devices requires changing that key, this protection scheme works well only when there is continuous network connectivity among all devices in the domain. Another drawback of this approach is that revoking devices part of the domain requires changing the domain key, which affects all devices.

The xCP architecture [3] has been proposed by IBM, and is based on broadcast encryption. The compliance checking protocol in xCP is based on the broadcast encryption algorithms introduced in [19]; because this only involves symmetric key operations, xCP compatible devices do not require hardware cryptographic accelerators, which is a great economical advantage. In this context, it is interesting to point out that our architecture accomplishes the same thing (it does not require cryptographic hardware accelerators on compliant devices), *without* making use of broadcast encryption and asymmetric cryptographic operations for intra-domain authentication.

The system model we introduce in this paper is based on the specification in [24]. However, [24] only lists a number of functional requirements and possible design options, without going into details about protocols and security mechanisms.

Finally, [22] describes an architecture supporting delegation of authorization to personal electronic devices used for electronic transactions. Although this is not directly related to content delivery for home networks, some of the delegation protocols described in [22] can be incorporated in "Authorized Domains" architectures.

7. CONCLUSION

We have described a security architecture for the "Authorized Domains" framework. Central to our design effort is a novel compliance checking protocol which allows individual device authentication in a loosely connected network environment, with minimal reliance on public key cryptographic operations. In addition to this, our architecture supports efficient and flexible revocation of compromised devices: revoking one compromised device in the domain, does not require performing a key update for all devices in the domain.

As for future work, recognizing the importance of consumer privacy, we intend to extend our architecture by incorporating privacy protection mechanisms, as suggested in [7].

8. ACKNOWLEDGMENTS

The authors would like to thank Peter Lenoir, Paul Koster, and all the anonymous reviewers for their constructive comments on this paper.

9. REFERENCES

[1] DVB - The Digital Video Broadcasting Consortium. http://www.dvb.org/.

[2] Secure Hash Standard. FIPS 180-1, Secure Hash Standard, NIST, US Dept. of Commerce, Washington D. C. April 1995.

[3] xCP Cluster Protocol. http://www.almaden.ibm.com/software/ds/ContentAssurance/ papers/xCP_DVB.pdf.

[4] Smartright technical white paper. http://www.smartright.org/images/SMR/content/SmartRight_ tech_whitepaper_jan28.pdf, Jan. 2003.

[5] C. Boyd. A Class of Flexible and Efficient Key Management Protocols. In *Proc. 9th IEEE Computer Security Foundation Workshop*, 1996.

[6] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.

[7] C. Conrado, F. Kamperman, G. J. Schrijen, and W. Jonker. Privacy in an Identity-based DRM System.

In *Proc. 14th Intl. Workshop on Database and Expert Systems Applications*, pages 389–395, Sept. 2003.

[8] B. Crispo, B. Popescu, and A. Tanenbaum. Symmetric key authentication services revisited. In *Proc. 9th Australasian Conference on Information Security and Privacy*, July 2004.

[9] Call for proposals for content protection & copy management technologies, July 2001.

[10] A. Eskicioglu and E. Delp. An overview of multimedia content protection in consumer electronic devices. *Signal Processing: Image Communication*, 16(5):681–699, April 2001.

[11] A. Eskicioglu, J. Town, and E. Delp. Security of Digital Entertainment Content from Creation to Consumption. *Signal Processing: Image Communication*, 18(4):237–262, April 2003.

[12] A. Fiat and M. Naor. Broadcast Encryption. In *Advances in Cryptology - CRYPTO '93*, pages 480–491, 1993.

[13] F.L.A.J.Kamperman and S.A.F.A.van den Heuvel and M.H.Verberkt. Digital Rights Management in Home Networks. In *Proc. IBC 2001*, pages 70–77, Sept. 2001.

[14] J. A. Halderman. Evaluating New Copy-Prevention Techniques for Audio CDs. In *Proc. 2002 ACM Workshop on Digital Rights Management*, 2002.

[15] J. Kohl and B. Neuman. The Kerberos Network Authentication Service (Version 5). Technical report, IETF Network Working Group, 1993. Internet Request for Comments RFC-1510.

[16] H. Krawczyk, M. Bellare, and R. Canetti. RFC 2104 - HMAC: Keyed-Hashing for Message Authentication. Internet RFC 2104, Feb. 1997.

[17] J. B. Lotspiech, S. Nusser, and F. Pestoni. Broadcast encryption's bright future. *IEEE Computer*, 35(1), 2002.

[18] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[19] D. Naor, M. Naor, and J. Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. In *Advances in Cryptology - CRYPTO '01*, pages 41–62, 2001.

[20] M. Ripley, C. Traw, S. Balogh, and M. Reed. Content Protection in the Digital Home. *Intel Technology Journal*, 6(9):49–56, 2002.

[21] B. Rosenblatt, B. Trippe, and S. Mooney. *Digital Rights Management, Business and Technology*. M&T Books, 2002.

[22] S. Sovio, N. Asokan, and K. Nyberg. Defining Authorization Domains Using Virtual Devices. In *SAINT Workshops 2003*, pages 331–336, 2003.

[23] S. G. Stubblebine and R. N. Wright. An Authentication Logic with Formal Semantics Supporting Synchronization, Revocation, and Recency. *IEEE Trans. Softw. Eng.*, 28(3):256–285, 2002.

[24] S. van den Heuvel, W. Jonker, F. Kamperman, and P. Lenoir. Secure Content Management in Authorized Domains. In *Proc. IBC 2002*, pages 467–474, Sept. 2002.

[25] C. Wong, M. Gouda, and S. Lam. Secure Group Communications Using Key Graphs. In *Proc. of the ACM SIGCOMM*, pages 68–79, 1998.

APPENDIX

A. A LOGICAL PROOF OF THE PROTOCOL INTRODUCED IN SECTION 4.2.2

In this section we examine the security of the device to device authentication protocol we introduced in Section 4.2.2. For our analysis we use the BAN [6] logic that has been extended as suggested by Wright and Stubblebine in [23] in order to formalize revocation and dealing with keyed hash functions. Both concepts are not present in the original BAN logic. We will not get into the details of this logic, assuming the reader is familiar with it.

We had to extend the original BAN logic with a new formula and two new postulates. The formula expresses the statement saying that a principal A checked the revocation list issued by S about key K and the key is not present in the list (thus K is valid)

Revocation

$$\neg(A \models \neg(S \models X))$$

Concerning the postulates, the first extend *once said* in *belief* if the statement has not been revoked after it has been uttered. The second states that assuming f a keyed hash function over any number of input, the key obtained by applying such a function on these inputs is trusted as long as one of the input is a trusted secret and one is a fresh nonce.

Revocation-check postulate

$$\frac{A \models S \sim X, \neg(A \models \neg(S \models X))}{A \models S \models X}$$

Key-derivation postulate

$$\frac{A \models A \overset{K}{\longleftrightarrow} B, A \models \#(N)}{A \models A \overset{f(\ldots,K,N,\cdot)}{\longleftrightarrow} B}$$

We start our analysis from the idealized version of the protocol, as required by the logic, recalling that the messages and parts of messages in cleartext are omitted, since they do not contribute to the logical properties of the protocol. The idealized protocol is shown in Figure 2, where S stands for the AD manager. $ID_{domain}, GDI_{device}$, and LDI_{device} are omitted because their purpose is to identify the sender and receiver of the message, and this indication is already captured by the specification of the principals (A and B) in the other constructs (i.e. $A \overset{K_{BA}}{\rightleftharpoons} B$).

(2) B $\longrightarrow$ A: $\{A \overset{K_{BA}}{\rightleftharpoons} B\}_{K_{AS}}$

(3) A $\longrightarrow$ B: $\{A \overset{K_{AB}}{\rightleftharpoons} B\}_{K_{BS}}, \{N_b, A \overset{K}{\longleftrightarrow} B\}_K$

(4) B $\longrightarrow$ A: $\{N_a, A \overset{K}{\longleftrightarrow} B\}_K$

Figure 2: Idealized device-to-device authentication protocol

The analysis consists of starting from assumptions that represent the beliefs of the parties when the run of the protocol starts and by applying the postulates of the logic verifying if the goal of authentication is achieved. This goal can

be expressed in term of beliefs of the two parties. Thus we might deem that authentication is complete between A and B if there is a K such that: $A \models A \overset{K}{\longleftrightarrow} B$, $B \models A \overset{K}{\longleftrightarrow} B$, $A \models B \models A \overset{K}{\longleftrightarrow} B$, and $B \models A \models A \overset{K}{\longleftrightarrow} B$.

The assumptions of the protocol are the following ones:

Assumptions

$$
\begin{array}{ll}
A \models A \overset{K_{AS}}{\longleftrightarrow} S & B \models B \overset{K_{BS}}{\longleftrightarrow} S \\
S \models A \overset{K_{AS}}{\longleftrightarrow} S & S \models B \overset{K_{BS}}{\longleftrightarrow} S \\
A \models A \overset{K_{AB}}{\rightleftharpoons} B & B \models A \overset{K_{BA}}{\rightleftharpoons} B \\
S \models A \overset{K_{AB}}{\rightleftharpoons} B & S \models A \overset{K_{BA}}{\rightleftharpoons} B \\
A \models (S \models\!\!> A \overset{K_{BA}}{\rightleftharpoons} B) & B \models (S \models\!\!> A \overset{K_{AB}}{\rightleftharpoons} B) \\
A \models (S \models\!\!> (B \mid\sim X)) & B \models (S \models\!\!> (A \mid\sim X)) \\
A \models \#(N_a) & B \models \#(N_b)
\end{array}
$$

The first four assumptions are about the shared keys between devices and the AD manager. The next four concern A's and B's authentication keys generated and assigned to them by the AD manager. The fact that the keys are unidirectional is represented by the fact that each device believes, directly, only its own directional key. The next two assumptions capture the belief of the client in the other party's directional authentication key via the AD manager, that has jurisdiction over these keys since he generates and assigns them. The next two assumptions indicates that devices trust the AD manager to forward a message from the other device honestly. This trust is used since the AD manager distributes the authentication keys and the fact that this is done off-line or on-line is not relevant. The last two assumptions show that two nonces are used and who considers them to be fresh.

Starting from the assumptions we can now proceed with the analysis. A receives message 2, so $A \triangleleft \{A \overset{K_{BA}}{\rightleftharpoons} B\}_{K_{AS}}$. A can decrypt the message because she knows and trusts K_{AS} thus, by applying the message-meaning, revocation check and jurisdiction postulates, we have:

$$\frac{A \models A \overset{K_{AS}}{\longleftrightarrow} S, A \triangleleft \{A \overset{K_{BA}}{\rightleftharpoons} B\}_{K_{AS}}}{A \models S \mid\sim A \overset{K_{BA}}{\rightleftharpoons} B}$$

$$\frac{A \models S \mid\sim A \overset{K_{BA}}{\rightleftharpoons} B, \neg(A \models \neg(S \models A \overset{K_{BA}}{\rightleftharpoons} B))}{A \models S \models A \overset{K_{BA}}{\rightleftharpoons} B}$$

$$\frac{A \models (S \models\!\!> A \overset{K_{BA}}{\rightleftharpoons} B), A \models S \models A \overset{K_{BA}}{\rightleftharpoons} B}{A \models A \overset{K_{BA}}{\rightleftharpoons} B}$$

Then B receive message 3 and he seees: $B \triangleleft \{A \overset{K_{AB}}{\rightleftharpoons} B\}_{K_{BS}}, \{N_b, A \overset{K}{\longleftrightarrow} B\}_K$.

For the first part of the message we can apply the same rule we just applied for A, thus by applying the message-meaning, revocation check and jurisdiction postulates on the fist part of the message B sees, we have:

$$B \models A \overset{K_{AB}}{\rightleftharpoons} B$$

Now we can apply the key derivation postulate, and we get the following:

$$\frac{B \models A \overset{K_{AB}}{\rightleftharpoons} B, B \models \#(N_b)}{B \models A \overset{K}{\leftrightarrow} B}$$

where $K = f(K_{AB}, K_{BA}, N_b, N_a)$.

By applying the message meaning and nonce-verification postulate on the second message received by B, we obtain:

$$\frac{B \models A \overset{K}{\leftrightarrow} B, B \triangleleft \{N_b, A \overset{K}{\longleftrightarrow} B\}_K}{B \models A \mid\sim (N_b, A \overset{K}{\longleftrightarrow} B)}$$

$$\frac{B \models \#(N_b), B \models A \mid\sim (N_b, A \overset{K}{\longleftrightarrow} B)}{B \models A \models A \overset{K}{\longleftrightarrow} B}$$

The protocol continues with B sending message 4 to A such that: $A \triangleleft \{N_a, A \overset{K}{\longleftrightarrow} B\}_K$. The analysis of this message is the same as the analysis of the second message received by B in step 3 of the protocol. Thus, by also applying the key-derivation, message-meaning and nonce-verification postulates we obtain:

$$A \models B \models A \overset{K}{\longleftrightarrow} B$$

We can conclude that the authentication goals are satisfied, since from the initial assumptions, the two participants reach the state where both of them believe they share a key they both trust. Additionally, each of participants believe the other party believes the same thing (strong authentication).

Since the limits of the BAN logic are well known, we recognize that a more challenging security analysis is also necessary. At the time of writing, we are working on a security analysis that use a more powerful model than the one assumed by the BAN logic (i.e. Dolev&Yao) for attacks and intruders and different tools (i.e. model checkers). However, due to space constraints, we will provide the full details of such analysis in a future paper.