

A New DRM Architecture With Strong Enforcement

Sascha Müller

Department of Computer Science

Technische Universität Darmstadt

Darmstadt, Germany

mueller@seceng.informatik.tu-darmstadt.de

Stefan Katzenbeisser

Department of Computer Science

Technische Universität Darmstadt

Darmstadt, Germany

katzenbeisser@seceng.informatik.tu-darmstadt.de

Abstract—We propose a new DRM architecture that utilizes a two-step enforcement process to enable strong security even in the case of a compromised DRM viewer. This is achieved by using novel cryptographic techniques of attribute-based encryption that make it possible to limit access to media to a subset of users that has to fulfill certain properties which are specified during the encryption process. We call these properties *static rules*. Static rules add an additional layer to the dynamic DRM enforcement framework that has to be overcome by potential attackers even if a DRM media operates in an unprotected environment. Finally, we demonstrate the practicability of this architecture by describing how static rules can be automatically extracted from licenses formulated in the standardized Open Digital Rights Language (ODRL).

I. INTRODUCTION

One of the major obstacles in the deployment of DRM schemes is the lack of available trusted viewers, which decrypt DRM-protected content and enforce a license at runtime. To date, no satisfactory solution for implementing a DRM viewer on a legacy operating system is known; the main reason seems to be the fact that the DRM client is usually executed in an untrusted environment which is under full control of an attacker and thus is exposed to possible unauthorized modifications that are very hard to detect and almost impossible to prevent. In this paper we propose a new architecture for DRM systems which allows to reduce the trust required in DRM viewers. To do this, we partition the set of rules into *static* and *dynamic* rules. Static rules are enforced by cryptographic means before an access to the media takes place, while dynamic rules must be enforced at runtime by the trusted viewer. This way, current cryptographic methods can be used to strongly enforce parts of DRM licenses. This is different to the approach deployed currently, where the enforcement of DRM rules depends entirely on a trusted software DRM viewer.

Our central contribution is the introduction of static rules. To give an intuition of this idea, consider a typical license of a DRM protected system where a media can only be accessed by a certain type of device. This is an example of a static rule: It must be checked before any kind of access to the media, no matter what specific usage (e.g., playing or copying) is desired. To statically enforce this rule, we can encrypt the media with a key that is only given to valid

devices. In fact, simple static rules like this are already in use: In most DRM systems, each media is encrypted with a *title key* K_T , which is sent encrypted along with the license and can only be decrypted by a private *player key* K_P which is known only to authenticated DRM viewers. This construction can be seen as a static rule stipulating that access must only be granted to an authenticated DRM viewer. If one manages to restrict usage of K_P to authorized viewers, trust relies upon the issuer of the key K_P , who – unlike the viewing platform – can be controlled by the media distributor. This observation is crucial to our approach.

Opposed to traditional DRM architectures, the media in our framework is not directly encrypted by a single title key, but with an arbitrary Boolean formula over a set of keys. In the most general case, these keys may even be maintained by different, independent authorities, and a DRM viewer has to retrieve the relevant keys from the respective authorities in order to decrypt the media. Each of these keys is associated with a certain property of the user or the platform that is executing the viewer and is only granted to the viewer if certain obligations are fulfilled. These obligations are part of the DRM license.

More formally, let m be a media and K_P the player specific key that is needed to access the media. We denote the title key that is needed to access m by K_T . Then the encrypted media can be distributed along with $E(K_P, K_T)$. Generalizing this traditional approach, we can use cryptography to enforce more properties. For example, let K_D be another key, that is only granted from a key-issuer after a certain date has passed. Using this approach, we can additionally encrypt the media with K_D to enforce that the media can only be accessed by an authorized viewer *and* after the given date:

$$E(K_D, E(K_P, K_T))$$

We generalize this idea so that any Boolean formula can be used. To achieve this, we use the notion of *attribute-based encryption* (ABE). In ABE schemes, users possess secret keys corresponding to sets of attributes that describe their properties (like group membership or actions he has taken). In *Ciphertext-Policy Attribute-Based Encryption* (CP-ABE),

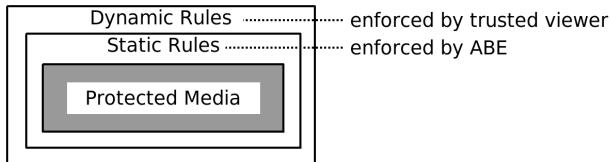


Figure 1. Structure of DRM rules.

a ciphertext is encrypted in a way that allows decryption only for users whose sets of attributes match a predicate $p(\mathbb{K})$, i.e., a Boolean formula, over the set of all attributes. In our example, where both K_D and K_P are required to decrypt K_T , the predicate could be written as $p(\mathbb{K}) := K_D \wedge K_P$, where $K_D, K_P \in \mathbb{K}$. The title key would then be encrypted as

$$E(p(\mathbb{K}), K_T).$$

In order to decrypt, a user has to acquire a set of attributes such that he is able to satisfy $p(\mathbb{K})$. We call the set of obligations that are enforced by such a predicate *static rules*.

Static rules add an additional layer to the dynamic DRM enforcement framework, supplementing the enforcement of the dynamic rules with cryptographic primitives. Figure 1 visualizes this relationship. Even if an attacker is able to overcome the enforcement of the dynamic rules, there are static rules that must be fulfilled before he is able to access the media. However, the player lacks cryptographic keys for this step. This is favourable as it allows a reduction of trust in the viewers: While classic DRM schemes are completely broken if an attacker manages to extract a secret player key, our scheme at least ensures that – even in case of a compromised viewer – some parts of the license, i.e., the static rules, are always enforced, since crucial keys are not available to the attacker.

It is, however, not very practical to require two policies (one containing the static rules and one containing the dynamic rules), and in fact, the distinction between static and dynamic rules might not be clear to license authors. Thus, we propose a *license conversion tool* as a central component of our framework. This conversion tool takes as input a DRM license and gives as output the enforceable sub-policy, containing only static rules which are suitable for use with a cryptographic algorithm (called attribute policy), and a slightly modified DRM license that marks the static rules, giving the DRM viewer information on what components are already enforced in the form of static rules. The attribute policy is enforced through a cryptographic construction, while the modified policy still needs to be enforced at runtime by the DRM viewer. This strategy effectively allows to reduce the trust in the viewer, as some important rules like group membership, payed fees, or a temporal usage range can be enforced cryptographically even on compromised DRM viewers.

To enforce rules we propose to use *Distributed Attribute-Based Encryption (DABE)* [1], which is a generalization of Ciphertext-Policy Attribute-Based Encryption [2]. In DABE, the secret attribute that make up the users' key rings can be distributed by an arbitrary number of independent *attribute authorities*, and users can claim their keys incrementally, while in CP-ABE schemes there is usually one party that gives a user a full set of secret attribute keys at once. Thus, DABE is very suitable for fine-grained, distributed access control and enforcement of static rules as proposed in our framework.

Similarly, in other scenarios, Broadcast Encryption (BE) is used to restrict access to media by cryptographic means (see, for example, [3]). In BE, a media is encrypted in a way that only certain subsets of users can decrypt. For this, the identities of all users has to be known to the encrypting party, and depending on the structure of the subset of eligible users, the ciphertexts may become very large. Attribute-Based Encryption (ABE) allows much more fine-grained access control, and the length of ciphertexts is independent on the concrete number of users of the system. However, there is some connection between BE and ABE. For example, BE has been used to add a revocation mechanism to ABE schemes [4].

The rest of the paper is structured as follows: In Section II we give some background on DABE. Subsequently we describe a DRM framework that utilizes DABE in Section III. To illustrate the practicability of our framework, we describe a conversion process of DRM licenses in a scenario where ODRL policies are used in Section IV. Finally, Section V concludes the paper.

II. DISTRIBUTED ATTRIBUTE-BASED ENCRYPTION

Ciphertext-Policy Attribute-Based Encryption (CP-ABE) allows to encrypt data under an access policy, specified as a logical combination of attributes. In [5] we described a construction that allows to incrementally claim private keys and support attributes maintained by different, independent attribute authorities, called *Distributed Attribute-Based Encryption (DABE)*. We will use DABE as it is more powerful than CP-ABE, and specially allows to incrementally add attributes to user keys at any time and even introduce new attributes after setup. Note that although the multi-authority case was discussed before [6]–[8], there is yet no ABE construction with a reduction proof that allows the incremental claim of attributes [6]. These properties are crucial in our scenario, but for many applications a CP-ABE construction might suffice. For a recent comparison of the features and differences of CP-ABE schemes, see [9].

We now describe the DABE scheme: The parties involved in a DABE scenario are a central trusted authority (which manages secret user keys), an arbitrary number of attribute authorities (each described by a unique identifier) which issue secret keys for attributes maintained by them, and an

arbitrary number of users who want to encrypt and/or decrypt data. The scheme consists of seven algorithms: *Setup*, *CreateUser*, *CreateAuthority*, *RequestAttributePK*, *RequestAttributeSK*, *Encrypt*, and *Decrypt*. The description of these algorithms is as follows:

Setup: The *Setup* algorithm takes as input the security parameter 1^k . It outputs the public key PK, which is used in all subsequent algorithms, and the secret master key MK. This algorithm is executed by a central authority.

CreateUser(PK, MK, u): The *CreateUser* algorithm takes as input the public key PK, the master key MK, and a user name u . It outputs a public user key PK_u that will be used by attribute authorities to issue secret attribute keys for u , and a secret user key SK_u , used for the decryption of ciphertexts. This algorithm is executed by the central authority once for each user.

CreateAuthority(PK, a): The *CreateAuthority* algorithm is executed by the attribute authority with identifier a once during initialization. It outputs a secret authority key SK_a .

RequestAttributePK(PK, \mathcal{A} , SK_a): The *RequestAttributePK* algorithm is executed by attribute authorities whenever they receive a request for a public attribute key. The algorithm checks whether the authority is responsible for the attribute \mathcal{A} . If this is the case, the algorithm outputs a public attribute key $PK_{\mathcal{A}}$ corresponding to \mathcal{A} , otherwise NULL.

RequestAttributeSK(PK, \mathcal{A} , SK_a , u , PK_u): The *RequestAttributeSK* algorithm is executed by the attribute authority with identifier a whenever it receives a request for a secret attribute key. The algorithm checks whether it has authority over attribute \mathcal{A} and whether user u with public key PK_u is eligible for \mathcal{A} . If this is the case, *RequestAttributeSK* outputs a secret attribute key $SK_{\mathcal{A},u}$ for user u . Otherwise, the algorithm outputs NULL.

Encrypt(PK, M , \mathbb{A} , $PK_{\mathcal{A}_1}, \dots, PK_{\mathcal{A}_N}$): The *Encrypt* algorithm takes as input the public key PK, a message M , an access policy \mathbb{A} and the public keys $PK_{\mathcal{A}_1}, \dots, PK_{\mathcal{A}_N}$ corresponding to all attributes occurring in the policy \mathbb{A} . The algorithm encrypts M with \mathbb{A} and outputs the ciphertext CT.

Decrypt(PK, CT, \mathbb{A} , $SK_u, SK_{\mathcal{A}_1,u}, \dots, SK_{\mathcal{A}_N,u}$): The *Decrypt* algorithm gets as input a ciphertext CT produced by the *Encrypt* algorithm, an access policy \mathbb{A} under which CT was encrypted, and a key ring $SK_u, SK_{\mathcal{A}_1,u}, \dots, SK_{\mathcal{A}_N,u}$ for user u , which includes the secret user key SK_u and secret attribute keys $SK_{\mathcal{A}_1,u}, \dots, SK_{\mathcal{A}_N,u}$ for attributes $\mathcal{A}_1, \dots, \mathcal{A}_N$ of user u . The algorithm *Decrypt* decrypts the ciphertext CT and outputs the corresponding plaintext M if the attributes $\mathcal{A}_1, \dots, \mathcal{A}_N$ were sufficient to satisfy \mathbb{A} ; otherwise it outputs NULL.

III. FRAMEWORK

To use DABE to strengthen the enforcement process, we propose the following DRM framework, depicted in Fig. 2. As usual, the media distributor formulates a license that he

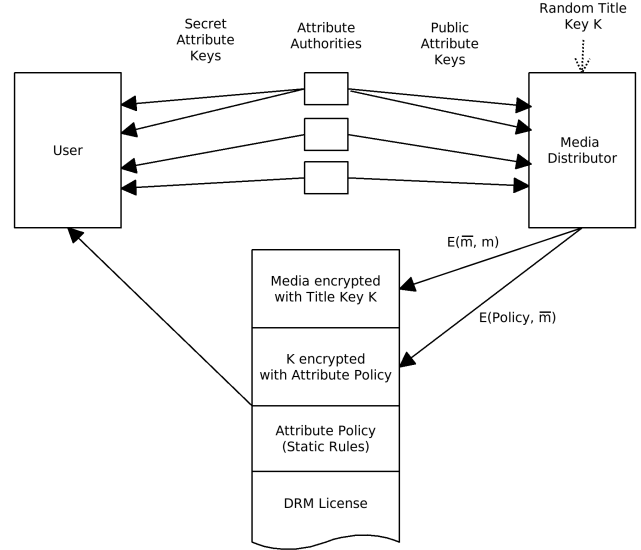


Figure 2. Involved parties, relationships, and data structures.

attaches to the media. This license includes both dynamic rules and static rules. In fact, from the point-of-view of the distributor there may not be any obvious differences between the two types of rules. Subsequently, he runs a conversion tool that takes as input the license and a set of *conversion rules*. The tool analyses the policy to find the components that contain static rules, and extracts an *attribute policy* \mathbb{A} (i.e., a sub-policy containing only the static rules) from the license. This attribute policy \mathbb{A} contains all cryptographically enforceable parts of the license and is used in the DABE encryption step. The media distributor furthermore obtains *public attribute keys* for all attributes referenced in \mathbb{A} from the respective attribute authorities. The media is then encrypted using a title key, which is in turn encrypted with the *Encrypt* algorithm of the DABE scheme. Thus, only users with secret attribute key rings that satisfy the attribute policy \mathbb{A} can decrypt the title key and the media. The media is distributed along with the encrypted title keys, the attribute policy \mathbb{A} and the modified ODRL expression.

Whenever a user downloads protected content, he inspects the attribute policy and obtains a set of secret attribute keys that fulfill the policy, unless he is already in possession of the required keys. He is able to access the title key by executing the *Decrypt* algorithm, which in turn can be used by the viewer to access the media.

Not shown in Fig. 2 is the central authority that initializes the global system parameters (algorithm *Setup*) and grants every user u a key pair (PK_u, SK_u) that is used by the attribute authorities to personalize the secret attribute keys. This is necessary to prevent collusion attacks, where sets of users try to combine their sets of secret attribute keys to collectively be able to access media that they can not access individually. Resistance against such attacks is provided

by all DABE and CP-ABE schemes. The global system parameters and public user keys created by this trusted authority are made available securely to all participants.

The task of the attribute authorities is the provision of attribute keys. At creation, each authority executes *CreateAuthority*, taking as input the global system parameters. Whenever queried for public attribute keys – usually by media distributors – the authority executes *RequestAttributePK*. All attribute keys are identified by strings, and each attribute authority is able to maintain an arbitrary number of attributes. Users may query the authorities for secret attribute keys. When this happens, the queried attribute authority has to verify if the user is eligible of the attribute, and if so, execute *RequestAttributeSK*, giving as input the public user key. The user may have to perform some actions (e.g. make a payment) before he becomes eligible for some of the keys.

Note that the attribute policy limits general access to the media by enforcing certain prerequisites for decryption, but it cannot control how the media is used once the attribute policy is fulfilled. Thus, only those rules are enforceable that specify conditions that must be fulfilled by the viewers before access is granted for the first time. Static rules are thus a subset of all rules of a policy. Dynamic rules will still be enforced at runtime by the DRM viewer.

IV. PROCESSING ODRL EXPRESSIONS

To illustrate the practicability of our framework, we consider a DRM scenario where the Open Digital Rights Language (ODRL) [10] is used to express DRM licenses. An ODRL expression allows to describe a set of actions a user is allowed to perform and a set of rules associated with each action. The rules describe properties that the user or the executing device has to fulfil before the associated actions can be taken. They are classified into constraints, requirements, and conditions. Constraints are limits to exercising the actions, requirements are obligations that must be fulfilled in order to be allowed to perform the action, and conditions specify exceptions that, if they become true, expire the permissions. We will subsume ODRL constraints and requirements as *rules* and further partition them into *static rules* and *dynamic rules*. For example, *static rules* could mandate that an action can only be performed by members of a certain group, on certain hardware, after a certain time has passed, or that the user must pay some fee before taking the action.

A. Encryption process

In detail, the encryption process, executed by media distributors, contains the following phases (see also Fig. 3 for a visualization of the process):

- 1) **XML Extraction.** This phase takes as input an ODRL expression Pol and a set of conversion rules that map cryptographically enforceable ODRL rules within Pol to a single attribute policy. It outputs a Boolean

expression in the form of a circuit containing a predicate $p(\mathbb{K})$ that describes the enforceable sub-policy of Pol . It also outputs a modified ODRL expression Pol' . In Pol' , all static attribute nodes are marked with a special XML attribute, so that the DRM viewer knows that they do not need to be enforced at runtime. However, we do propose to keep all original rules intact, so that a client still sees from the ODRL policy his terms of use.

- 2) **Attribute Expansion.** The Boolean expression created by the XML extraction phase may contain numerical and date/time attributes and numerical comparisons. As CP-ABE (and DABE) constructions usually support only Boolean attributes and comparisons, these numerical attributes need to be suitably encoded, see Section IV-D.
- 3) **Policy Finalization.** The final steps are dependent on the DABE construction that is used. The tree obtained in the last phase is optimized (e.g. by combining adjacent AND/OR nodes) and converted to the required access structure format (i.e., DNF [5], [11], [12], access trees [2], [13]–[15], or share-generating matrices of linear secret-sharing schemes [16]). Call the result \mathbb{A} .
- 4) **Encryption.** The media distributor retrieves public keys for all attributes used in \mathbb{A} from the respective attribute authorities. In a multi-authority scenario he might have a choice between different, independent authorities, and the media distributor can modify the conversion rules of the XML extraction phase to constitute his decision of the authorities he trusts most. However, it must be clear from the names of the attributes in the attribute policy which attribute authority was chosen, so that the users can request secret attribute keys from the same authorities. The distributor executes the algorithm $\text{Encrypt}(\text{PK}, K_T, \mathbb{A}, \text{PK}_{A_1}, \dots, \text{PK}_{A_N})$ with the random title key K_T and the public attribute keys as input and receives as output a ciphertext CT . Subsequently, he encrypts the plaintext media m with the title key K_T and publishes the encrypted media together with K_T , the license and access policy:

$$\langle E(K_T, m), CT, \mathbb{A}, Pol' \rangle.$$

Note that like in classic DRM implementations, the static rules of Pol' should also be enforced by the DRM viewer that controls usage of the media at runtime. For example, let there be two keys K_A and K_B , each associated with an attribute. A policy might rule that – for instance – playing a media, the attribute associated with K_A is required and for copying the media, the attribute of K_B is needed. This could be turned into a predicate $p(\mathbb{K}) := K_A \vee K_B$ which can be enforced statically. With this predicate, the DRM

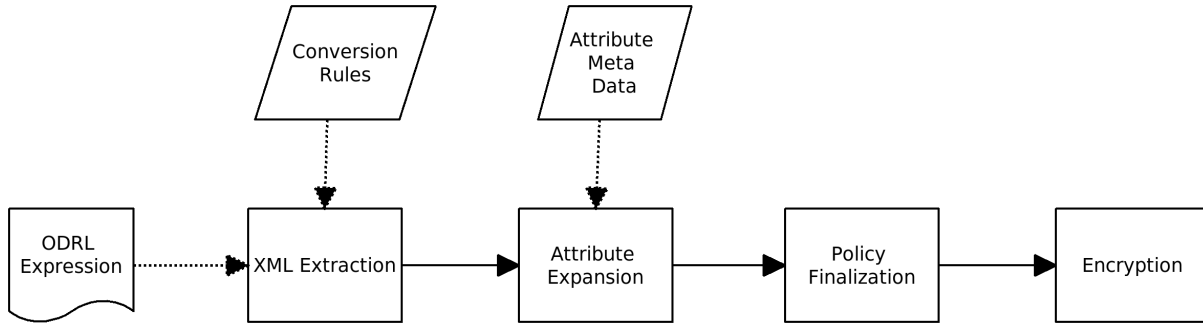


Figure 3. ODRL expression conversion process.

viewer is assured that the media can only be accessed if the user possesses either K_A or K_B , but when a specific usage is requested, it still needs to verify that the user has the right attribute, so the rule is enforced by the trusted viewer just like dynamic rules. However, by the approach in this paper, the sub-policy describing the static rule set (i.e., the predicate $p(\mathbb{K})$) is *also* enforced cryptographically. Thus, enforcing static rules both cryptographically and dynamically allows for very fine-grained policies that constrain specific actions by specific static and dynamic rule sets.

B. Parsing Agreements

An ODRL expression consists of offers and agreements; agreements are used as DRM licenses and are thus the only components that have to be enforced at runtime. Agreements are XML blocks enclosed in `<agreement> ...</agreement>` tags. An agreement consists of a set of permissions, where a set of rules can be associated with each permission. Figure 4 shows an example permission that allows printing of a document once an amount US\$ 20 has been paid in advance. This rule is an example of a static rule, as an attribute rule

```
media1234.hasPaidFor.USD >= 20
```

can be added to the attribute policy \mathbb{A} , where 1234 is the identifier of the media. For permissions with multiple rules, all rules must be honored, so the enforceable rules in the attribute policy must be combined with the AND operator.

ODRL supports the declaration of various permissions inside a single permissions block, corresponding to different rule sets associated with different actions that can be performed alternatively. We can not control the type of action that a user performs once the decryption has taken place, but obviously he has to fulfil at least one rule set to be able to decrypt. Thus we allow decryption of the title key if *any* of the rule sets is satisfied (all other parts of the license must be enforced at runtime). To achieve this, the rule sets of different permissions are connected by an OR operator to produce the policy \mathbb{A} . Formally, given a set of permissions P_i , where each permission contains enforceable rules $R_{i,j}$, the generated attribute policy has the form

```
<permission>
  <print>
    <requirement>
      <prepay>
        <payment>
          <amount currency="USD">20</amount>
        </payment>
      </prepay>
    </requirement>
  </print>
</permission>
```

Figure 4. Sample permission with rule

$$\bigvee_i \bigwedge_j R_{i,j}$$

Note, however, that some $R_{i,j}$ could themselves become Boolean expressions after the Attribute Expansion phase (see Section IV-D), so the final access policy might not yet be in DNF.

C. Path Conversion

A conversion rule maps an ODRL rule to an attribute. We automate the extraction of rules from ODRL permissions by conversion rules.

Consider, for example, the rule shown in Fig. 4. The ODRL rule can be written in its XPath form as `permission/print/requirement/prepay/payment/amount[@currency="USD"]/20`. To find all rules of this form within an ODRL expression, we can use a regular expression: `requirement/prepay/payment/amount[@currency="(.)"]/(.+)`. This expression will match all ODRL rules of the type `prepay`, and output two *match groups*: One that represents the currency as a string (“USD” in our example), and one that represents the amount (20 in our example). We can use these match groups to automatically create the above rule in the attribute policy.

ODRL Name	Attribute
Individual	user.<context>
Group	user.groupmember.<context>
CPU	device.cpu.<context>
Network	device.network.<context> device.network.ip
Screen	device.screen.<context>
Storage	device.storage.<context>
Memory	device.memory.<context>
Printer	device.printer.<context>
Hardware	device.hardware.<context>
Software	device.software.<context>
Date Time	datepassed.<year> datepassed.<year>-<month> datepassed.<year>-<month>-<day> datepassed.numerical
Prepay	media<id>.hasPayedFor.<currency>
Accept	media<id>.accepted
Register	media<id>.registered

Table I
STATIC RULES OF THE ODRL STANDARD WITH STANDARDIZED
ATTRIBUTE MAPPINGS.

D. Representing ODRL rules by Attributes

In this section we show how enforceable ODRL rules, as encoded by the extraction process described above, are represented as attributes. Enforceable attributes deal with properties of the viewer, as his identity, affiliation, age, role, and group membership, that the user trying to access the media must possess. Alternatively they could be connected to the document itself, like payments made for purchase. Table I lists all ODRL rules that can be expressed as static rules.

In our sample implementation we use a systematic naming scheme where attribute names incorporate URLs of attribute authorities, components describing ODRL context blocks, and unique identifiers.

The names also support attributes that can represent integers. For many scenarios it might be enough to use Boolean attributes to represent numerical values. For example, if a license requires the payment of US\$ 20, inclusion of an attribute `media{id}.hasPayedFor.USD20` is enough to enforce the rule. However, ODRL allows very flexible rules that may, for example, restrict the number of pages that a user can read depending on the concrete amount of money that he has paid. Thus, the attributes should be represented in a way to allow numerical comparisons within the attribute policy \mathbb{A} . We encode integers as “bags of bits”, as described in [2].

Attributes that deal with date or time can be encoded in various ways. There could be Boolean attributes that are only issued after some point in time has passed (this approach is similar to using a key K_D as mentioned in the introductory example), or the time could be encoded numerically.

```
<?xml version="1.0" encoding="UTF-8"?>
<o-ex:rights xmlns:o-ex="http://odrl.net/1.1/ODRL-EX
  xmlns:o-dd="http://odrl.net/1.1/ODRL-DD">
<o-ex:agreement>
  <o-ex:asset>
    <o-ex:context>
      <o-dd:uid>samplemedia</o-dd:uid>
      <o-dd:name>A Sample Media</o-dd:name>
    </o-ex:context>
  </o-ex:asset>
  <o-ex:permission>
    <o-dd:play>
      <o-ex:constraint>
        <o-dd:group>
          <o-ex:context>
            <o-dd:uid>samplegroup</o-dd:uid>
          </o-ex:context>
        </o-dd:group>
      </o-ex:constraint>
    </o-dd:play>
    <o-dd:print>
      <o-ex:requirement>
        <o-dd:prepay>
          <o-dd:payment>
            <o-dd:amount o-dd:currency="USD">
              20
            </o-dd:amount>
          </o-dd:payment>
        </o-dd:prepay>
      </o-ex:requirement>
    </o-dd:print>
  </o-ex:permission>
</o-ex:agreement>
</o-ex:rights>
```

Figure 5. Example ODRL expression.

E. Example

Fig. 5 shows a complete ODRL expression, and Fig. 6 shows the resulting attribute policy after XML Extraction (reformatted for better readability). Note that the second permission (the `<o-dd:print>` block) is the same as the example of Fig. 4, which was already discussed. Here, the media ID was taken from the `<asset>` block near the beginning of the ODRL expression. In the Attribute Expansion phase, the two numeric comparisons (`datepassed.numerical >= 20011231` and `mediasamplemedia.hasPayedFor.USD >= 20`) will be expanded to their bags of bits representation.

As proof of concept we implemented the crucial parts of the policy conversion framework in the Python 3 programming language. Our tool takes as input an ODRL policy and a set of conversion rules as described in the preceding sections. It converts the XML file to its DOM representation and parses it, listing all rules contained in its agreement blocks, and identifying which of them are enforceable according to the given conversion rules. In the next step it applies the conversion rules and outputs all

```

(
(
user.groupmember.samplegroup AND
(
datepassed.2001-12-31 OR
datepassed.numerical >= 20011231
)
) OR
mediasamplemedia.hasPaidFor.USD >= 20
)

```

Figure 6. Attribute policy after XML Extraction phase.

resulting attribute rules. The tool is also able to convert comparison rules of numerical attributes with constants to comparisons with “bag of bits” representations. More work on this implementation is currently done, as we are aiming for a practical and usable DRM framework that is able to automatically identify all static rules, extract Boolean attribute policies from ODRL expressions, and enforce them using a DABE construction.

V. CONCLUSION

In this paper, we showed how the trust required in DRM viewers can be reduced by using Distributed Attribute-Based Encryption. We have identified those ODRL expressions that can be cryptographically enforced through CP-ABE or DABE and proposed a framework that automatically extracts those enforceable policies from ODRL expressions as Boolean attribute policies and encodes them. This allows to implement a DRM scheme where an enforceable attribute policy is automatically extracted from an ODRL expression. This attribute policy can in turn be used as input to a DABE scheme which encrypts the title key with the policy. This ensures that, even when a viewer is compromised, static rules of the ODRL expressions are enforced.

REFERENCES

- [1] S. Müller, S. Katzenbeisser, and C. Eckert, “On multi-authority ciphertext-policy attribute-based encryption,” *Bulletin of the Korean Mathematical Society (B-KMS)*, vol. 46, no. 4, pp. 803–819, July 2009, to appear.
- [2] J. Bethencourt, A. Sahai, and B. Waters, “Ciphertext-policy attribute-based encryption,” in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2007, pp. 321–334.
- [3] A. Fiat and M. Naor, “Broadcast encryption,” in *CRYPTO*, ser. Lecture Notes in Computer Science, D. R. Stinson, Ed., vol. 773. Springer, 1993, pp. 480–491.
- [4] N. Attrapadung and H. Imai, “Conjunctive broadcast and attribute-based encryption,” in *Pairing*, ser. Lecture Notes in Computer Science, H. Shacham and B. Waters, Eds., vol. 5671. Springer, 2009, pp. 248–265.
- [5] S. Müller, S. Katzenbeisser, and C. Eckert, “Distributed attribute-based encryption,” in *11th International Conference on Information Security and Cryptography, ICISC 2008*, ser. Lecture Notes in Computer Science, P. J. Lee and J. H. Cheon, Eds., vol. 5461. Springer, 2008, pp. 20–36.
- [6] M. Chase and S. S. Chow, “Improving privacy and security in multi-authority attribute-based encryption,” in *16th ACM Conference on Computer and Communications Security, CCS 2009*, 2009, to appear.
- [7] M. Chase, “Multi-authority attribute based encryption,” in *Theory of Cryptography Conference, TCC 2007*, ser. Lecture Notes in Computer Science, S. P. Vadhan, Ed., vol. 4392. Amsterdam, The Netherlands: Springer, February 2007, pp. 515–534.
- [8] H. Lin, Z. Cao, X. Liang, and J. Shao, “Secure threshold multi authority attribute based encryption without a central authority,” in *9th International Conference on Cryptology in India, INDOCRYPT 2008*, ser. Lecture Notes in Computer Science, D. R. Chowdhury, V. Rijmen, and A. Das, Eds., vol. 5365. Springer, 2008, pp. 426–436.
- [9] T. Nishide, K. Yoneyama, and K. Ohta, “Attribute-based encryption with partially hidden ciphertext policies,” *IEICE Transactions*, vol. 92-A, no. 1, pp. 22–32, 2009.
- [10] “Open digital rights language,” August 2002. [Online]. Available: <http://odrl.net>
- [11] A. Kapadia, P. P. Tsang, and S. W. Smith, “Attribute-based publishing with hidden credentials and hidden policies,” in *Proceedings of The 14th Annual Network and Distributed System Security Symposium (NDSS)*, Mar 2007, pp. 179–192.
- [12] L. Cheung and C. C. Newport, “Provably secure ciphertext policy ABE,” in *ACM Conference on Computer and Communications Security*, P. Ning, S. D. C. di Vimercati, and P. F. Syverson, Eds. ACM, 2007, pp. 456–465.
- [13] R. Bobba, H. Khurana, and M. Prabhakaran, “Attribute-sets: A practically motivated enhancement to attribute-based encryption,” in *14th European Symposium on Research in Computer Security, ESORICS 2009*, ser. Lecture Notes in Computer Science, M. Backes and P. Ning, Eds., vol. 5789. Springer, 2009, pp. 587–604.
- [14] V. Goyal, A. Jain, O. Pandey, and A. Sahai, “Bounded ciphertext policy attribute based encryption,” in *35th International Colloquium on Automata, Languages and Programming, ICALP 2008*, 2008.
- [15] X. Liang, Z. Cao, H. Lin, and D. Xing, “Provably secure and efficient bounded ciphertext policy attribute based encryption,” in *4th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2009*, W. Li, W. Susilo, U. K. Tupakula, R. Safavi-Naini, and V. Varadarajan, Eds. ACM, 2009, pp. 343–352.
- [16] B. Waters, “Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization,” SRI International, Tech. Rep., 2008, work in progress. [Online]. Available: <http://eprint.iacr.org/2008/290.pdf>