EXPERIMENTS IN RIGHTS CONTROL

EXPRESSION AND ENFORCEMENT

Cheun Ngen Chong
张俊年

Members of the dissertation committee:

prof. dr. P. H. Hartel	University of Twente (promoter)
dr. S. Etalle	University of Twente (assistant promoter)
prof. dr. W. Jonker	University of Twente
prof. ir. E. Michiels	University of Twente
prof. dr. W. G. Vree	Delft University of Technology
dr. ir. W. B. Teeuw	Telematica Instituut
prof. dr. P. M. G. Apers	University of Twente (chair)

University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands.
Internet: http://www.utwente.nl
This thesis is included in the CTIT Ph.D.-thesis Series.
ISSN 1381-3617.
Internet: http://www.ctit.utwente.nl

This thesis is published in the Telematica Instituut Fundamental Research Series.
ISSN 1388-1795.
Telematica Instituut, P.O. Box 589, 7500 AN Enschede, The Netherlands
E-mail: info@telin.nl
Internet: http://www.telin.nl

This thesis is also published in the IPA Dissertation Series.
Eindhoven University of Technology, Room HG 7.22, Den Dolech 2, Eindhoven, The Netherlands.
Internet: http://www.win.tue.nl/ipa/

This thesis was edited with Vim and typeset with LaTeX.
Keywords: rights control, digital rights management, license, rights expression language, rights enforcement, tamper-resistance

Cover Design	Studio Oude Vrielink, Losser and Jos Hendrix, Groningen
Book Design	Lidwien van de Wijngaert and Henri ter Hofte
Printing	Universal Press, Veenendaal, The Netherlands

CTIT Ph.D.-Thesis Series Number: 05-68
IPA Series Number: UT.2005-03
TI Series Number: 013
ISBN 90-75176-40-6 TI/FRS/013

EXPERIMENTS IN RIGHTS CONTROL

EXPRESSION AND ENFORCEMENT

DISSERTATION

to obtain
the doctor's degree at the University of Twente,
on the authority of the rector magnificus,
prof. dr. W. H. M. Zijm,
on account of the decision of the graduation committee,
to be publicly defended
on Friday, February 4, 2005 at 13.15

by

Cheun Ngen Chong

born on 25 March 1977,

in Kuala Lumpur, Malaysia

This dissertation is approved by:

prof. dr. Pieter H Hartel (promoter) and
dr. Sandro Etalle (assistant-promoter)

For my family
In memory of my dearest grandmother...

ABSTRACT

The Internet has transformed our long-term perception on working, entertainment, and living rapidly. We can now work comfortably in our own home, shop for our groceries without stepping outside the house, and enjoy high-quality entertaining digital content, such as music or film. This digital content can easily be produced and copied with available digital technologies; and the content can be distributed and shared through the Internet almost effortlessly. This phenomenon has created a wide variety of usage scenarios, and has also induced huge loss to the film and music industry through piracy. To solve this problem, we have to protect the digital content by controlling how the users are using the content. Thus, *rights control* has emerged as a potential solution.

A myriad of technical, social, and legal questions related to rights control have appeared. For instance: How can we control the way in which digital content is used? How can we guarantee that users abide by our rules when using the digital content? And, how can we allow users to use the content anyway they like without violating our rules?

This thesis addresses the aforementioned questions by performing a series of experiments, which consist of (1) studying available and novel usage *scenarios*; (2) deriving the *requirements* of rights control from the scenarios; (3) proposing the *design* of a language and its implementation architecture for rights control based on the requirements; (4) building *prototypes* that serve as proof-of-concept for aspects of the design; and finally

(5) performing various *evaluations* on these prototypes to justify the practicality of the design.

Several contributions are made from two aspects, namely rights expression and rights enforcement:

An experimental logic-based rights expression language (REL), namely LicenseScript is proposed. LicenseScript is based on logic programming and multiset rewriting. The language has a simple but elegant syntax, which provides fine-grained control over the digital content; and, it has rich formal semantics, which can be used to verify useful properties.

- A wide variety of usage scenarios are studied and implemented in LicenseScript. A thorough comparison of LicenseScript with other available XML-based RELs is conducted in the usage scenarios. This shows that LicenseScript is highly expressive and flexible.

- Fair Use (United States Codes, U.S.C, Section 107 Title 17 Chapter 1, Fair Use Doctrine) is modelled in LicenseScript. Fair Use is useful in expressing the end-user's usage rights, which are legal from the copyright perspective, but might not allowed by the content providers initially. This shows another advantage of LicenseScript.

A rights enforcement architecture that implements LicenseScript is proposed. This architecture is composed of several security modules achieving different objectives that support rights control:

- An experimental digital rights management (DRM) system, which implements rights control is built. The system is able to associate the user's identity, and other security attributes, e.g. user's role to the rights of using the digital content. A prototype is created and its performance is evaluated. An analysis is provided using a SPIN model.

- A LicenseScript license interpreter is designed and built by using state-of-the-art software tools. It is able to interpret, execute and alter the LicenseScript licenses.

- A secure audit logging protocol is proposed, which refines Schneier and Kelsey's protocol by incorporating a tamper-resistant token.

The protocol is able to detect unauthorized deletion, manipulation or creation of the audit logs. The protocol has been implemented in a real system, and its performance has been assessed.

- A license protection scheme based on a key tree and a tamper-resistant hardware token is proposed. The scheme is able to protect and enforce different usage rights on different parts of a license or content flexibly. This scheme has been implemented in a prototype. Several tests have been done on the prototype to adjust the practicality, and the protocol has been formally verified using CoProVe.

- An experimental streaming audio protection approach, i.e., StreamTo is proposed. The StreamTo player is able to play the protected audio piecemeal. A hardware token decrypts the protected audio frame by frame. Two prototypes based on two different hardware tokens are built, and their performance is evaluated and compared.

SAMENVATTING

Het internet heeft onze lange-termijn visie op werk, entertainment en leven snel veranderd. We kunnen nu comfortabel thuis werken, onze boodschappen doen zonder de deur uit te hoeven, en genieten van hoogwaardig digitale aanbod zoals muziek of film. Dit digitaal aanbod kan gemakkelijk worden geproduceerd en gekopieerd met beschikbare digitale technieken; en de inhoud kan moeiteloos gedistribueerd en gedeeld worden via internet. Dit fenomeen heeft gezorgd voor een grote diversiteit aan gebruikssscenarios, en heeft tevens een groot verlies aan inkomsten veroorzaakt bij de film- en muziekindustrie door piraterij. Om dit probleem op te lossen moeten we het digitale aanbod beschermen door te controleren hoe de gebruikers de inhoud gebruiken. Zodanig is controle van rechten als een mogelijke oplossing naar voren gekomen.

Een scala technische, sociale en juridische vraagstukken gerelateerd aan de controle van rechten zijn ontstaan. Bijvoorbeeld: hoe kunnen we de manier waarop digitaal aanbod wordt gebruikt controleren? Hoe kunnen we garanderen dat gebruikers de regels respecteren van ons digitale aanbod? Hoe kunnen we de gebruikers toestaan het digitale aanbod naar eigen believen te gebruiken zonder dat zij onze regels overtreden?

Dit proefschrift behandelt de bovengenoemde vragen door middel van het uitvoeren van een aantal experimenten, die bestaan uit (1) een studie van beschikbare en “novel usage” scenarios; (2) het afleiden van vereisten voor controle van rechten vanuit de scenarios; (3) het voorstellen van

een taalontwerp en de implementatie van een architectuur voor controle van rechten op basis van de vereisten; (4) het bouwen van prototypes die dienen als "proof-of-concept" voor de ontwerpen; en tot slot (5) het uitvoeren van verschillende evaluaties van deze prototypes om het praktisch aspect van het ontwerp te rechtvaardigen.

De bijdragen van dit proefschrijft zijn te verdelen in twee categorieën, namelijk "rechten uitdrukking" en "rechten naleving":

Een experimentele op logica gebaseerde "rights expression" taal (REL), namelijk LicenceScript, wordt voorgesteld. LicenceScript is gebaseerd op logisch programmeren en "multiset" herschrijven. De taal heeft een eenvoudige doch elegante syntaxis, die kan zorgen voor een verfijnde controle op het digitale aanbod; en hij heeft een rijke formele semantiek, die gebruikt kan worden om een bruikbare eigenschappen te controleren.

- Een grote variëteit aan gebruiksscenarios wordt bestudeerd en gëimplementeerd in LicenceScript. Een grondige vergelijking tussen LicenceScript en andere beschikbare op XML gebaseerde REL wordt verwerkt in de gebruiksscenarios. Dit laat zien dat LicenceScript uitermate expressief en flexibel is.

- Een bruikbare copyright wet, namelijk "Fair Use" (United States Codes, U.S.C, Section 107 Title 17 Chapter 1, Fair Use Doctrine) wordt behandeld in LicenceScript. Fair Use is handig in het uitdrukken van de eindgebruikers gebruiksrechten, dat legaal is vanuit het perspectief van copyright maar initieel misschien niet is toegestaan door de contentaanbieders. Dit toont een ander voordeel van LicenceScript aan.

Een architectuur gericht op het naleven van rechten, die LicenceScript implementeerd wordt voorgesteld. Deze architectuur wordt samengesteld uit verschillende veiligheidsmodules die resulteren in verschillende doelstellingen die controle op rechten ondersteunen:

- Een experimenteel Digital Rights Management (DRM) systeem, dat rechten controle implementeerd, wordt gebouwd. Het systeem is in staat de gebruikers identiteit, en andere veiligheidsonderdelen zoals de gebruikers rol, te koppelen aan het recht om de digitale aanbod te

gebruiken. Een prototype wordt gecreerd en de performance gevalueerd. Een analyse wordt aangeboden door gebruik van een SPIN model.

- Een LicenceScript vergunning interpreter is gecreëerd en gebouwd door gebruik te maken van state-of-the-art software middelen. Deze is in staat de LicenceScript vergunning te interpreteren, uit te voeren en te veranderen.

- Een beveiligde "audit logging protocol" wordt voorgesteld, dat Schneier en Kelsey's protocol verfijnd door het incorporeren van een "tamper-resistant" token. Het protocol is in staat ongeautoriseerd verwijderen, manipulatie of creatie van de "audit logs" te detecteren. Het protocol is geïmplementeerd in een echt systeem en de performance is beoordeeld.

- Een beschermingsschema voor de license wordt voorgesteld gebaseerd op een "key tree" en "tamper-resistant hardware token". Het schema is in staat verschillende gebruiksrechten op verschillende delen van een license of aanbod flexibel te beschermen. Het schema is geïmplementeerd in een prototype. Diverse tests zijn op het prototype uitgevoerd om het praktische aspect aan te passen en het protocol is formeel geverifieerd aan de hand van CoProVe.

- Een experimentele benadering van de bescherming van streaming audio, nl. StreamTo, wordt voorgesteld. De StreamTo-player is in staat de beveiligde audio in stukken (packets) af te spelen. Een hardware-token ontcijfert de beveiligde audio frame voor frame. Twee prototypes gebaseerd op twee verschillende hardware-tokens worden gebouwd, en de performance wordt geëvalueerd en vergeleken.

ACKNOWLEDGEMENTS

Finally, I am here, finishing my own thesis. The typos and grammar mistakes later discovered in the thesis will only be treated as special features of my thesis.

I consider myself extremely lucky for this opportunity of working in the academic and (partly) industrial area simultaneously, despite the fact that a Ph.D. is sometimes too pressurizing. I feel that I have grown a lot in these 4 years, technically and mentally. Not to mention that I have improved physically too, thanks to the habitual gymnastic, fitness and swimming schedule, which I follow almost daily to relax from the stress. What makes me most grateful is that I am actually surrounded by friends and colleagues. Their kindness, support, and constant help makes me feel less alone fighting for the Ph.D. I would like to name a few of them here.

First of all, I would like to show my deepest gratitude to my daily supervisor, Pieter Hartel. He carried out his supervision with great dedication. His optimism, enthusiasm, and problem-solving capability never stopped to amaze me. His help and support is really what motivates me. He reads everything I have written, and corrects every mistake. His speed and thoroughness is what we, the Ph.D. students are always fascinated by. He is a busy professor, yet his door is always open for his students. Furthermore, he never feels tired after a busy working day, and step into the Ph.D. student's room asking, "anything you need my help with". I actually have learned a lot from him in handling social interaction with other

members as for teamwork. I am very grateful he did not give up on me, when I had almost given up on myself.

Sandro Etalle, who is my second supervisor, has been helping me constantly to alleviate my daily worries and stress. He is extremely busy everyday, yet he does not mind spending evening time, after a 10-hour busy working hours, to discuss my progress. His gentle and humble manner – he is never reluctant to apologize whenever he has made a mistake (which rarely happens) – incentivizes me to work harder.

Yee Wei Law has been my house-mate for nearly 4 years (we actually have known each other for almost 8 years now). My "don't-care" attitude in the house must have bothered him a lot, yet he seldom makes any much complaints. On the other hand, he has done all the cleaning and management of the house. What amazes me the most is his enthusiasm in learning new stuff, and his never-tired attitude. Furthermore, his patience in sharing his knowledge with me is unbelievable.

Ricardo Corin always supports me in a subtle way. His intelligence is what I admire the most, as well as his industrious manner. He never hesitates to share his intellectual knowledge with me. I certainly have learned a lot in doing research and handling some social issues from him. More importantly, I certainly appreciate and will miss his sense of humour, which always makes the office full of laughter.

Nikolay Kavaldjiev has inspired me a lot in handling some difficulties in a daily life. He can handle most emergencies without any sweat. Furthermore, he always accompanies me to the daily gymnastic and fitness activities. Talking to him about my project sometimes can clear up the mist in my mind.

Vasughi Sundramoorthy is my fellow Malaysian colleague (same as Y. W. Law). She is a funny and caring friend, with a lot of energy. We always talk about the plans for our own future, and it is quite inspiring.

Gabriele Lenzini has become my office room-mate since September 2004. He is really a great room-mate. He is quiet, considerate, and extremely patient – he has never made any complaints to whatever I have done that might have bothered him. He does not mind at all sharing his experience in life with me.

Jeroen Doumen has been extremely helpful to me. Even if we have been colleagues for only one year, he is never reluctant to provide all kinds

of valuable help.

René van Buuren and other Telematic Institute members have contributed to the SUMMER and LicenseScript projects. René, especially, has given me unexpected support not only in the projects, but also in my Ph.D., at a time when I had doubts about beginning the Ph.D.

Geert Kleinhuis, Nico Zornig, Igor Passchier, Rieks Joosten and other TNO Telecom members have been very helpful in working with me on the RGE project (Geert has also helped me in the SUMMER project). Without them, it would have been impossible for me to finish my project smoothly.

Lodewijk Smit has shared his experience and tricks in writing the thesis with me. Thus, he has helped me a lot in planning and writing the thesis.

Marlous Weghorst is our secretary. She has helped me a lot in the bureaucratic matters, such as filling in forms, booking flight tickets for conferences, and other non-trivial daily business. Her helpful and cheerful mood always makes the office more a home.

Ruth Griepink has been helping me with my English for about 6 months. Her patience and guidance have improved my English writing skill for writing this thesis.

Frederik Zuidberg always listens to my complaints and problems patiently. He is really one of the most important friends whom I must thank wholeheartedly. His tireless support has really helped me get through some of the depressing and disappointing moments.

Michael Soon Xin Ng is my friend from University of Southampton. He has become my mentor since the first day we have known each other. His indomitable spirit always reminds me never to give up because there is always a way.

Peter Henderson, from the University of Southampton provided me various support to start my Ph.D. at Southampton.

Lajos Hanzo, also from University of Southampton has supervised my Bachelor final year project. His patient guidance has inspired my interest in doing a Ph.D.

I thank my high-school old-time friends, namely Lee Lan Lim, Yong Peng Cham, Kim Tee Koh, Lee Kun Tay, and Poh Ying Hip for their hospitality when I spend my vacation back in Malaysia; and, their support when I worked in Netherlands. Also, I want to thank the MSc students

who have been working hard for me. They are Zhonghong Peng, Bin Ren and Jieyin Cheng.

Moki provides me with what I need to work at full speed everyday. She never complains when I ask for more spirit-mover from her. She is quite, kind and beautiful. Frankly, without her I would not have enough energy to work throughout the day. She is a lovely coffee machine we have in our office.

Last but not least, I would like to share my success with my family in Malaysia. My parents have given me all the freedom to decide what I liked to do in my future. Without their support, I would not have finished the Ph.D.. I would like to thank my grandmother especially. I always regret that I could not see her before she passed away when I was in my third year undergraduate degree. She asked my parents to hide her illness from me on the phone because she knew I was having my final examinations. Without her unselfish love, I would never imagine that I could reach that far in my life.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I
PROLOGUE

The thesis explores the design and implementation of the rights expression language LicenseScript. A rights expression language (REL) is intended to express usage rights on digital content. The implementation interprets the language and carries out the actual rights enforcement.

Extensive research has been conducted to make the rights expression language flexible, expressive and capable of providing fine-grained control on the digital content. However, the current rights expression languages are still limited in their ability to express dynamic features of rights control. LicenseScript presents a major advance in this respect. We will dwell on this in part II (pp.29).

The implementation of rights expression language presents many interesting problems of its own. For example, the extent to which hardware tamper-resistance is required. This will be addressed in part III (pp.95).

This part of the thesis provides an introduction to the thesis: aims and objectives, goals, scope and results.

Aim and Objectives

Digital rights management (DRM) is one of the most prominent applications of rights control. DRM is a collection of technologies, which allow the players on the content distribution value chain, e.g. content provider to restrict the usage of the content in various way. The content provider and the content consumer (end-user) are at opposite ends of the value chain. There maybe more players involved, such as the content creator, content publisher, content distributor, etc.

DRM raises a lot of questions, such as, how can the intellectual property rights of the content owned by the creator be protected, how can the benefits of all the players be taken care of on the content value chain, how can a different business model generate revenue to the content provider, distributor and other players on the value chain, etc.

The research questions we try to answer in the thesis are: How can we *control the way* in which digital content is used? How can we *guarantee that the users abide by our rules* (rules can be made by any player involved on a value chain, depending on the business models) when using the digital content? And, how can we *allow the users to use the content anyway they like* without violating our rules?

The main aim of the thesis is finding the answers to these questions, which we intend to achieve by pursuing the following objectives:

1. To provide an in-depth study of the two aspects of rights control, i.e., rights expression and rights enforcement.
2. To design a flexible and expressive rights expression language.
3. To implement the language with a selection of rights enforcement mechanisms based on using a hardware token (personal computers are completely open and unable to offer sufficient protection).

Scope

Digital rights management is an experimental subject. No proven technology exists even though there are many, incompatible, proprietary solutions. Most of these, if not all, are broken.

Therefore, we choose to perform *experiments* with novel language and implementation features. In our experiments, we describe:

- **Scenarios**, showing why flexibility in rights expression is essential.
- **Requirements**, which are derived from the scenarios to give a detailed account of language and implementation features.
- **Design**, proposing the language and implementation, which is based on the requirements. We include formal analysis when appropriate.
- **Prototypes**, which serve as a proof-of-concept for aspects of the design.
- **Evaluations**, which are performed on the prototypes, to show the applicability of the design.

Results

The main results of the thesis are:

1. An experimental logic-based rights expression language: LicenseScript. We show by comparing LicenseScript to its competitors that LicenseScript is able to capture a large variety of digital content usage scenarios.

2. Several experimental security mechanisms to enforce the rights have been implemented. These include audit logging, license storage and streaming audio protection. In these security mechanisms, we make use of a hardware token.

3. For each of the security mechanisms, we provide an evaluation to show the appropriateness of the mechanisms to rights control.

Projects

The work described in the thesis has taken place in three industrial projects (the black boxes) and three MSc projects (the dark gray boxes), as shown in Figure 1:

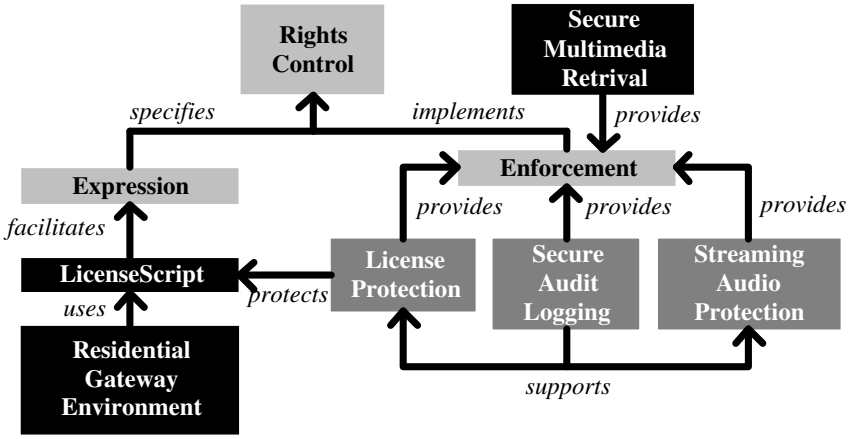

Figure 1: The structure of the thesis, shown as an entity-relationship diagram.

- **Secure Multimedia Retrieval**: SUMMER was a Dutch national project carried out with BTS-subsidy of the Ministry of Economic Affairs. It ran from April 2000 until September 2002. The partners of the SUMMER project include KPN Research, Center for Telematics and Information Technology (CTIT) and Department of Electrical Engineering of the University of Twente, Stichting V2_Lab, AVV Department of the Ministry of Transport, and CWI National Research Institute for Mathematics and Computer Science. The SUMMER project focused on secure, effective and efficient retrieval of multimedia data. Our part of the project focused on the implementation of rights control, which will be elaborated in chapter 5.

- **LicenseScript**: LicenseScript was partially funded by Telematica Institute. The project ran from October 2002 to October 2004. The goal of the LicenseScript project is to develop and demonstrate an integrated framework for analysis and design of secure information delivery system. Specifically, this project aims at the development of a framework in which one can specify, analyze and enforce usage rights. This will be elaborated in chapter 2.

- **Residential Gateway Environment** (RGE): The main objective of this project is to perform research concerning intelligent Residential Gateways. The project ran from March 2003 until February 2004. In future, consumers at home can have access to a great diversity of services via many different telecommunication networks by using an integrated, open and transparent gateway. KPN Research (now known as TNO Telecom) together with Philips Research, the technical universities of Eindhoven, Delft and Twente are performing the project. Our part of the project is to use LicenseScript in the RGE to facilitate service management and rights control. This is discussed in chapter 6.

- **MSc Projects** (Hardware Token Application): We have supervised three MSc projects, each of which contributes to part of the thesis. The main objectives of these projects are to develop security mechanisms to protect the license and the content, and providing enforcement support to rights control:

– **Secure Audit Logging**: Audit logging is intended to record all relevant actions on content, such as play or copy. Secure audit logging is based on a set of protocols that are able to detect unauthorized modification, fabrication, and deletion of audit logs. Secure audit logging provides fundamental support for the security of license protection and streaming audio protection. This is discussed in chapter 7.

– **License Protection**: A license carries usage rights and keys to unlock the protected content. Therefore, the license requires adequate protection. In this project, a license protection scheme for LicenseScript is proposed. It is able to ensure the confidentiality and integrity of the license or parts thereof. This protection scheme also works with a hardware token. This is discussed in chapter 8.

– **Streaming Audio Protection**: Content requires protection so that rights enforcement can be realized. In this project, streaming media is our focus. Similar to the two previous MSc projects, in this project a protection scheme using a hardware token is proposed. This is discussed in chapter 9.

Organization

In this section, we show how the thesis is divided into four parts. Each part is composed of several chapters, each of which has been published in a conference/workshop or a technical report.

In this (first) part of the thesis, there is one chapter:

- Chapter 1 (pp.11) provides the background to the thesis, i.e., a categorization of access control, usage control and rights control.

The remaining parts are also composed of several chapters:

- Part II (pp. 29) discusses rights expression languages. We propose an experimental logic-based REL, namely LicenseScript. Three chapters form this part"

- Chapter 2 (pp.35) introduces the rights expression language LicenseScript, which is based on multiset rewriting and logic programming.
- Chapter 3 (pp.53) compares two XML-based RELs, namely XrML and ODRL, with LicenseScript by exploring a number of usage scenarios. We show that LicenseScript is more flexible and expressive than XML-based RELs.
- Chapter 4 (pp.77) shows that LicenseScript is able to model fair use, as described by the United States Codes (U.S.C) Fair Use Doctrine. Fair use allows end-users to use the content in a manner that is not restricted by the content providers, as long as the usage purpose is *non-profit*.

- Part III (pp. 95) discusses several experimental security mechanisms that are needed by rights enforcement. We use a hardware token to achieve audit logs, license protection and streaming media protection. Three chapters form this part:

  - Chapter 5 (pp.101) provides a more detailed explanation of the background by describing a digital rights management (DRM) system. DRM is an application of rights control. In this chapter, identification, authentication and authorization (IAA) for rights control is addressed. Our DRM system has been built in the context of the SUMMER project.
  - Chapter 6 (pp.123) discusses an implementation of LicenseScript for the management of services. This is done in the context of the RGE project.
  - Chapter 7 (pp.143) discusses how we implement Schneier and Kelsey's secure audit logging protocol by using a hardware token. Secure audit logging can detect unauthorized tampering with audit logs.
  - Chapter 8 (pp.157) explains how we can store LicenseScript licenses in a hardware token. We can ensure the confidentiality and integrity of the licenses.

  - Chapter 9 (pp.181) explains how we can protect streaming audio, e.g. MP3 or WMA by using a hardware token. Thereby, we can enforce usage rights on the streaming audio.

- Part IV (pp. 203) summarizes the thesis, provides concluding remarks and future research possibilities.

Publications

To conclude, in this section, we list our refereed conference and workshop publications that constitute the core of the thesis, and further unrefereed CTIT technical reports (most of which are submitted for publication).

Refereed Conference/Workshop Contributions

1 C. N. Chong, S. Etalle, P. H. Hartel, R. Joosten, and G. Kleinhuis. Service Brokerage with Prolog. *7th International Conference on Enterprise Information Systems (ICEIS 2005*), 2005, pages To appear. INSTICC Press.
(`http://www.ub.utwente.nl/webdocs/ctit/1/000000f2.pdf`)

2 C. N. Chong, B. Ren, J. Doumen, S. Etalle, P. H. Hartel, and R. Corin, License Protection with a Tamper-Resistant Token. *5th Workshop on Information Security Applications (WISA 2004)*, volume 3325 of *Lecture Notes in Computer Science*, 2004, pages 224–238. Springer-Verlag.
(`http://www.ub.utwente.nl/webdocs/ctit/1/000000ff.pdf`)

3 C. N. Chong, S. Etalle, P. H. Hartel, and Y. W. Law. Approximating Fair Use in LicenseScript. In T. M. T. Sembok, H. B. Zaman, H. Chen, S. R. Urs, and S. H. Myaeng, editors, *6th International Conference of Asian Digital Libraries (ICADL'2003)*, volume 2911 of *Lecture Notes in Computer Science*, 2003, pages 432–443, Springer-Verlag.
(`http://www.ub.utwente.nl/webdocs/ctit/1/000000cf.pdf`)

4 C. N. Chong, S. Etalle, and P. H. Hartel. Comparing Logic-based and XML-based Rights Expression Languages. In R. Meersman and

Z. Tari, editors, *Proceedings of On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops*, volume 2889 of *Lecture Notes in Computer Science*, 2003, pages 779–792, Springer-Verlag. (http://www.ub.utwente.nl/webdocs/ctit/1/000000cd.pdf)

5 C. N. Chong, R. Corin, S. Etalle, P. H. Hartel, W. Jonker, and Y. W. Law. LicenseScript: A novel digital rights language and its semantics. In K. Ng, C. Busch, and P. Nesi, editors, *3rd International Conference on Web Delivering of Music (WEDELMUSIC)*, 2003, pages 122–129, IEEE Computer Society Press.
(http://www.ub.utwente.nl/webdocs/ctit/1/000000bf.pdf)

6 C. N. Chong, Z. Peng, and P. H. Hartel. Secure audit logging with tamper-resistant hardware. In D. Gritzalis, S. D. C. di Vimercati, P. Samarati, and S. K. Katsikas, editors, *18th IFIP International Information Security Conference (IFIPSEC)*, volume 250 of *IFIP Conference Proceedings*, 2003, pages 73–84. Kluwer Academic Publishers.
(http://www.ub.utwente.nl/webdocs/ctit/1/00000099.pdf)

7 C. N. Chong, R. van Buuren, P. H. Hartel, and G. Kleinhuis. Security attribute based digital rights management (SABDRM). In F. Boavida, E. Monteiro, and J. Orvalho, editors, *Joint Int. Workshop on Interactive Distributed Multimedia Systems/Protocols for Multimedia Systems (IDMS/PROMS)*, volume 2515 of *Lecture Notes in Computer Science*, 2002, pages 339–352. Springer-Verlag.
(http://www.ub.utwente.nl/webdocs/ctit/1/00000079.pdf)

8 Y. W. Law, C. N. Chong, S. Etalle, P. H. Hartel and R. Corin. Licensing Structured Data with Ease. In *Int. Workshop for Technology, Economy, Social and Legal Aspects of Virtual Goods*, 2003, page paper 11.
(http://www.ub.utwente.nl/webdocs/ctit/1/000000d0.pdf)

9 C. N. Chong, R. Corin, S. Etalle, P. H. Hartel, and Y. W. Law. LicenseScript: A Novel Digital Rights Language. In *Int. Workshop for Technology, Economy, Social and Legal Aspects of Virtual Goods*,

2004, page paper 11.
(http://www.ub.utwente.nl/webdocs/ctit/1/000000ba.pdf)

CTIT Technical Reports

10 J. Cheng, C. N. Chong, J. Doumen, S. Etalle, P. H. Hartel, and S. Nikolaus. StreamTo: Streaming Content using Tamper-Resistant Tokens. Technical report TR-CTIT-04-47, 18 pages, Nov. 2004, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, 2004.

11 C. N. Chong, S. Etalle, P. H. Hartel, R. Joosten, and G. Kleinhuis. Inter-library Service Brokerage in LicenseScript. Technical report TR-CTIT-04-33, 10 pages, Jul. 2004, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, 2004.
(http://www.ub.utwente.nl/webdocs/ctit/1/00000104.pdf)

12 R. Corin, C. N. Chong, S. Etalle, and P. H. Hartel. How to pay in LicenseScript. Technical Report TR-CTIT-03-31, 5 pages, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, 2003.
(http://www.ub.utwente.nl/webdocs/ctit/1/000000ce.pdf)

CHAPTER 1

INTRODUCTION

The Internet has revolutionized our daily life. It has transformed the way we communicate (e.g. chat, weblog, email, and messenger), live (e.g. online shopping, Internet banking), work (e.g. small office home office, e-commerce, e-business), and play (e.g. massive multiplayer role playing games, world wide web). More advanced networking protocols, more power-efficient and lighter hardware with bigger memory and faster processors are developed everyday to meet increasing demands from just under a billion users for digital content.

Digital content is anything that can be created, stored, processed, managed and distributed by using the digital technology. Digital content can be categorized into fancy media (games, film, video or music) and textual media (ebook, document, data, etc). Digital content can be rendered in high quality, is compact in size. Digital volatile disc (DVD), compact disc (CD) and MPEG-1 audio layer 3 (MP3) have almost entirely replaced our traditional video and audio tapes/records.

The Internet and digital content have a profound impact the way business is done. For instance, by using Windows Media Player 9, we can convert the music tracks from a CD to MP3, with only a couple of clicks. Then, with the help of a peer to peer (P2P) file sharing mechanism [Kalker et al., 2004] (Napster and Kazza) or BitTorrent [Cohen, 2003], countless copies of these MP3's can be distributed worldwide almost instantly.

Therefore, the intellectual property rights of the content are constantly being threatened by illegal copying and distribution of the content [Hartung and Ramme, 2000].

Digital content also needs to be protected from being misused. For example, hospitals need to protect specific parts of a patient's medical record from the insurance company but share these parts with other hospitals; parents want to protect their underage children from unsuitable content but they want to encourage them to watch educational content.

To solve these problems, we need to *control the usage rights on the content*. A usage right (digital right or simply right) is a permission granted by a content provider to a user to perform a particular operation on a content. Rights include rights for direct use of objects (such as read), delegation of rights (such as permission to trade), and rights for administering access (such as modify subject and object attributes that in turn determine rights). Rights can be divided into many functional categories [Rosenblatt et al., 2002], such as render rights (e.g. read), transport rights (e.g. move), etc.

The development of rights control is still in its adolescence. Therefore, the thesis presents an experimental approach focusing on the language aspect of rights control.

In this section, we present a taxonomy showing that rights control is related to the classical subject of access control. Section 1.1 explains access control. Section 1.2 dwells on usage control, which Sandhu and Park describe as the next vision of access control [Sandhu and Park, 2003]. Section 1.3 provides a brief overview of rights control, which represents the main contribution of the thesis. Section 1.5 summarizes the introduction.

1.1 Access Control

To log in to a computer system a valid username and password are needed. We authenticate ourselves to protect our files from others. Similarly, we need to present our bank card and enter a valid PIN number to withdraw money from ATM machines to protect our money.

Those are just some examples of authentication, which is a necessary prerequisite to access control. The purpose of authentication is to ensure

that only legitimate users enter a system. Once the user is authenticated, the system allows the user to perform certain operations on the system resources corresponding to their authorizations. Following the examples given above, with the valid username and password, the user *can use* the computer to perform her daily tasks; with the valid bank card and PIN number, the user *can withdraw* a certain amount of cash from the machine.

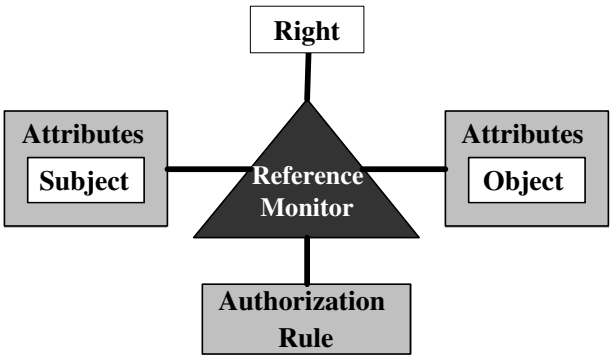

Figure 1.1: The fundamental model of access control, showing that the reference monitor decides the rights on the object for the subject. The decision depends on the attributes of the subjects and objects, and authorization rules.

The purpose of access control is to restrict the operations that a legitimate user can perform on the system resources [Sandhu and Samarati, 1994]. Access control normally involves two entities, namely a *subject* and an *object*, each of which are further characterized by *attributes*. The fundamental model of access control of Lampson [1974], is shown in Figure 1.1:

- The reference monitor decides and enforces the rights the subjects possess on the objects. The usage decision is made corresponding to the authorization rules and the attributes of the subject and object.
- A subject is normally an active entity, which initiates a request to perform an operation on an object. For example, user.
- An object is normally a passive entity, which is used and asked (to be used) by the subject. For example, file.

- A right is a permission for the subject to perform a specific operation on the object. For example, read.
- An attribute is a property of a subject and an object, which is used to decide if the access request can be granted. For example, the certificate identity of the subject.
- An authorization rule is a requirement that must be satisfied to allow the subject to perform an operation on the object. For example, the subject's identity must be on the system database.

1.1.1 Mechanisms

Having introduced the main concepts involved in access control, we will now consider the three main implementation mechanisms:

- Mandatory Access Control (MAC) defines a security level for the objects in the system. To access these objects, the user must possess the corresponding security level. This stems for the military domain.
- Discretionary Access Control (DAC) restricts access to objects based on the user's identity and/or groups to which the object belongs. This access control is discretionary in the sense that a subject with a certain access permission is capable of passing that permission (direct or indirect) to another subject.
- Role-based Access Control (RBAC) associates the access permissions to roles, which are associated with the users administratively. RBAC is proposed to overcome the inflexibility of MAC and DAC [Sandhu and Samarati, 1994]. The main difference is that RBAC uses roles rather than identities.

1.2 Usage Control

Access control is not sufficiently powerful to support rights control on digital content because it only deals with the authorization request and decision once, i.e., at the time the request is made. It does not support

continuous controls for long-lived access, which is one of the features required by the usage control on digital content. For example,

Example 1. A rented video can only be viewed for the next 3 hours.

Additionally, access control does not take into consideration the change of state of the subject and object before and after of the access, i.e., *mutability* of the subject and object attributes. For example,

Example 2. After rented a video, the credit of the subject is deducted with the value of the video rental.

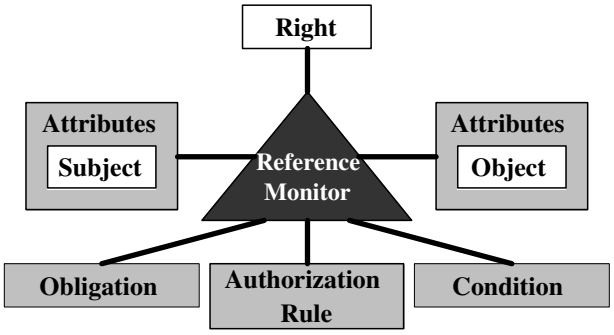

Figure 1.2: The usage control model components.

To control rights on digital content, two additional decision factors (in addition to the authorization rules) are introduced [Sandhu and Park, 2003], as shown in Figure 1.2:

- An *obligation* is a requirement imposed on the subject (before or after) using the object. An obligation is an operation that a subject ought to perform. For instance,

  Example 3. The user must *pay* before playing a music track.

- A *condition* is a constraint-based requirement that the subject and object must meet before or after the usage rights are granted. For example,

Example 4. The user is only allowed to play the film *within one year after she purchases the film*.

A condition is one of the decision factors that the system should verify during the authorization process before granting the permission to the subject. This is distinct from the authorization rules in the sense that the conditions can be dynamic, i.e., the states of the permissions can be altered with time.

	Description	Application
preA	Usage decision is performed before the access and no attributes are updated.	Traditional access control.
onA	Usage request is always allowed and checked repeatedly for continuous access.	Restricted number of times to access a content.
preB	Obligations have to be fulfilled before the access is permitted.	Register before obtaining content.
onB	Obligations need to be fulfilled continuously before the access is granted.	Must click on the advertisement for 10 times before entering the Web site.
preC	Environmental or time-based restrictions must be met before accessing.	Check the validity of the rights before using the content.
onC	Conditions are checked continuously while the rights are being exercised.	System status changes suddenly and may have to terminate the usage of content.

Table 1.1: The descriptions and applications of the basic UCON models, corresponding to the decision factors (Authorizations, oBligations, and Conditions) and continuity of usage control (pre and ongoing).

Sandhu and Park [2003] propose a family of usage control models (the UCON family). Table 1.1 briefly describes these models (the column Description) and provides an example of application of the corresponding

model (the column Application). These models cover the areas of traditional access control, trust management and digital rights management (DRM) (which we will discuss later chapter 5).

Example 1 and Example 4 are examples of the preC model; Example 3 is an example of the preB model. Example 2 is explained further in section 1.2.1.

1.2.1 UCON$_{\mathrm{ABC}}$

Park and Sandhu [2004] have further classified the usage control models presented in Table 1.1 into 16 models based on an additional criteria: mutability, which allows updates on subject or object attributes at different times. If all attributes are immutable, this implies that no updates are possible. With mutable attributes, updates are plausible before (pre), during (ongoing) or after (post) the right is exercised.

These 16 models are called the UCON$_{\mathrm{ABC}}$ models. They are indicated by the '✓' (checked entries) in Table 1.2. The unchecked entries indicate the cases where updates are not likely to be useful in practice. Example 2 is an example of "preA post-update" model. The reader may refer to Park and Sandhu's article for more details [Park and Sandhu, 2004].

	immutable	pre-update	ongoing-update	post-update
preA	✓	✓		✓
onA	✓	✓	✓	✓
preB	✓	✓		✓
onB	✓	✓	✓	✓
preC	✓			
onC	✓			

Table 1.2: The 16 basic UCON$_{\mathrm{ABC}}$ models.

To illustrate the implementation of the UCON$_{\mathrm{ABC}}$ models later, we reproduce here a summary of the formalization of UCON$_{\mathrm{ABC}}$:

- S and O represents the subject and object, respectively.
- $ATT(S)$ and $ATT(O)$ represent the set of attributes of the subject and object, respectively.

- Two basic predicates are defined for making the usage decisions:

$$\begin{aligned} allowed(s, o, r) &\implies conditions \\ stopped(s, o, r) &\impliedby conditions \end{aligned}$$

  These predicates indicate that the subject s is allowed respectively that any previous allowance has been stopped to exercise right r on the object o, depending on whether the $conditions$ are satisfied.

To complete the discussion of the UCON_{ABC} model, we show an example here, taken from Sandhu and Park's article [Park and Sandhu, 2004]:

Example 5. DRM pay-per-use with a prepaid credit.

$$\begin{aligned} M & \text{ is a set of monetary amounts} \\ credit: & \; S \rightarrow M \\ value: & \; O \times R \rightarrow M \\ ATT(s): & \; \{credit\} \\ ATT(o, r): & \; \{value\} \\ allowed(s, o, r) \Rightarrow & \; credit(s) \geq value(o, r) \\ preUpdate(credit(s)): & \; credit(s) = credit(s) - value(o, r) \end{aligned}$$

Here $credit$ is a function that returns the amount of money that the subject (s) has; $value$ is a function that returns the monetary value of the rights (r) to be exercised on the object (o); and $preUpdate$ is a procedure to perform update operations on the attributes.

The interpretation of the Example 5 is as follows: if the credit of a subject is not less than the value of the requested usage, the request is allowed. Once the request is allowed, the subject's credit is deducted by the value of the usage.

The UCON_{ABC} models do not show any enforcement mechanisms, or how the updates on the mutable attributes have to be performed. The models also do not include implementation details. Our work does provide these details.

As shown by Sandhu and Park, the UCON_{ABC} model is a comprehensive and conceptual framework that is able to model traditional access control as well as more modern and sophisticated usage control on the content.

1.3 Rights Control

The rights in the usage control models discussed above do not have attributes, which means that rights are immutable. Therefore, it is not possible to implement some useful usage scenarios. For example,

Example 6. Alice obtains an educational video from her teacher, who allows her to watch it and then to give the video to Bob so that he can watch it also.

UCON_{ABC} cannot express such usage scenario because UCON_{ABC} cannot express a right that Alice can exercise, and which allows her to update right attributes. It will in turn allow Bob to watch the video. Therefore, we propose rights control as a further extension of usage control. In this section, we explain the features of rights control, and then provide a glimpse of LicenseScript, which is able to model rights control.

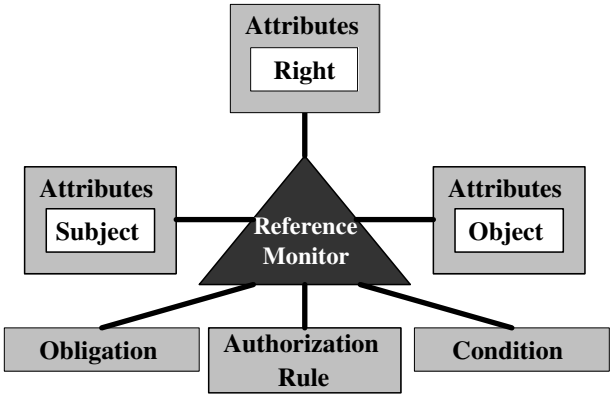

Figure 1.3: The rights control model.

In rights control, rights, similar to subjects and objects, have mutable attributes. Rights can also be updated, just like the subjects and objects. The rights control model, as shown in Figure 1.3, extends the usage control model of Figure 1.2 by associating attributes to the right.

This allows us to model: (1) *derivative rights* (Rosenblatt et al. [2002] propose four other types of digital rights, namely *render rights*, *transport rights*,and *utility rights*, which can be captured by the rights control model,

as will be discussed later in the thesis), (2) *administrative rights*, (3) *delegation rights*, and (4) *reverse rights* [Park and Sandhu, 2004]. These rights are related to each other. Example 6 is an example of administrative right and delegation right because Alice can modify the attributes to allow Bob to watch the video.

A *derivative right* comes in two flavours: (1) a right to reuse or reproduce an object that may produce another object, e.g. clip a music to produce another music, and (2) a right that after being exercised on an object will create a secondary object, e.g. play a music that creates a log. In first case, the secondary object may inherit some attributes of the rights of the original object (depending on the business model or provider's control). The second case is closely related to the reverse right, in the sense that a secondary object, which is controlled by the reverse right, is created by exercising the derivative right.

An *administrative right* is a right that can (1) manipulate another right, e.g. to create a new right in addition to the existing rights, or (2) administer another right, e.g. to modify object and subject attributes that in turn determine the right. In first case, attributes of the rights are modified.

A *delegation right* transfers a right to another entity, e.g. to sell the right to play a music.

A *reverse right* allows a beneficiary (e.g. end-user) to initiate rights control on the secondary object created after exercising a derivative right, e.g. to control the access of the content provider on the user's payment information after playing a film. The beneficiary should be able to manipulate attributes of the rights of the secondary object.

Rights control supports complex scenarios. Unfortunately, this makes rights control itself also less than simple. To explore the consequences of the additional complexity, we had to experiment with (1) scenarios, (2) expressiveness of the language used to describe rights control, and (3) the implementation of rights enforcement.

1.3.1 LicenseScript

LicenseScript [Chong et al., 2003a] is the embodiment of rights control. In this section, we will provide a glimpse of LicenseScript but defer the complete description to part II (chapter 2 pp. 35).

LicenseScript is based on logic programming and multiset rewriting. The basic construct is the license, which has the following form:

$$\text{object\_name}(Content, Clauses, Bindings)$$

Here object_name is the name of the LicenseScript object; $Content$ is the unique identifier of the associated content; $Clauses$ is a list of Prolog clauses that decide if the operations requested are allowed or forbidden; and $Bindings$ is a list of attributes that carry the status of the LicenseScript object.

A clause has the following form:

```
head :- atom_1,atom_2,...,atom_n.
```

Here `head` is the head of the clause (i.e., the name and parameters of the clause), and the conjunction of `atom_1,...,atom_n` is the body of the clause. Authorization rules, obligations, conditions and mutability are captured by the clauses.

LicenseScript licenses are notionally gathered in a multiset on which multiset rewrite rules operate. These capture aspects of communication and updates. The rewrite rules take the following form:

$$\begin{array}{rcl} rule\_name(arguments) & : & multiset_1 \longrightarrow multiset_2 \\ & \Longleftarrow & conditions \end{array}$$

Here $rule\_name$ is the name for the rule; $arguments$ are the arguments for this rule; $multiset_1$ and $multiset_2$ refer to the multiset of before and after the execution of the rule, respectively; and $conditions$ must be satisfied for the rule to apply. The conditions invoke queries over the clauses.

To illustrate that LicenseScript can model rights control, we use Example 6 (More examples will be shown in part II). We define a LicenseScript license for the educational video:

$$\text{license}(video, Clauses, Bindings)$$

The list $Clauses$ contains some clauses that allow Alice to watch the video and allow others (e.g. Bob) to watch it with permission of Alice:

```
canwatch(Sub,B1,B2) :-
   get_value(B1,watch_rights_holders,S),
   is_member(Sub,S).
candelegate(Sub1,Sub2,B1,B2) :-
   get_value(B1,watch_rights_holders,S),
   is_member(Sub1,S),
   add_member(Sub2,S,S2),
   set_value(B1,watch_rigths_holders,S2,B2).
```

Here, `get_value(X,Y,Z)` retrieves the value of binding `X` from list `Y` and stores the value to variable `Z`; and `set_value(W,X,Y,Z)` sets the value of binding `X` from list `W` to value `Y` and stores it to list `Z`; `is_member(X,Y)` checks if `X` belongs to list `Y`; and `add_member(X,Y,Y2)` adds `X` to list `Y`, returning list `Y2`.

The list $Bindings$ contains the following bindings:

```
watch_rights_holders=[alice]
```

Here, `rights_holders` contains a list of authorized subjects, in this case `alice`.

With this license, Alice can exercise the delegation right by executing the multiset rewrite rule $delegate(\texttt{alice}, \texttt{bob}, \texttt{video})$:

$$\begin{array}{lcl} \texttt{delegate}(S1, S2, O) & : & \text{license}(O, Cs, Bs_1) \rightarrow \text{license}(O, Cs, Bs_2) \\ & \Leftarrow & Cs \vdash candelegate(S1, S2, Bs_1, Bs_2) \end{array}$$

Here, $Cs \vdash candelegate(S1, S2, Bs_1, Bs_2)$ is a condition of the rule $delegate(\cdot)$, which basically means that execution of the query $candelegate$ $(S1, S2, Bs_1, Bs_2)$ in the clauses Cs yields *success*.

After Alice has executed $delegate(\cdot)$, the binding `rights_holders` has been updated by adding Bob's identity `bob`:

```
watch_rights_holders=[alice,bob]
```

This indicates that with the updated license, Bob can execute the multiset rewrite rule $watch(\texttt{bob}, \texttt{video})$:

$$\begin{array}{lcl} watch(S, O) & : & \text{license}(O, Cs, Bs_1) \rightarrow \text{license}(O, Cs, Bs_2) \\ & \Leftarrow & Cs \vdash canwatch(S, Bs_1, Bs_2) \end{array}$$

1.3.2 Implementing UCON$_{\mathrm{ABC}}$ in LicenseScript

In this section, we implement Example 5 of UCON$_{\mathrm{ABC}}$ in LicenseScript to illustrate the difference between UCON$_{\mathrm{ABC}}$ and LicenseScript:

$$\mathrm{license}(object, Clauses, Bindings)$$

Here, $object$ is the representation of the object. From Example 5, the pre-authorization of using the $object$ is that the subject must have enough credit for paying the value of performing the right; and the post-update is that the subject's credit is deducted by the value of performing the right.

We use clause `canallow` to capture the pre-authorization and post-update:

```
canallow(Subject,Right,B1,B2) :-
    get_value(B1,subject,S),
    Subject==S,                 %authentication
    get_value(B1,credit,M),
    get_value(B1,Right,V),
    M >= V,                     %pre-Authorization
    NewM = M - V,               %post-Update
    set_value(B1,credit,NewM,B2).
```

We use several bindings to capture the necessary attributes of the subject and object:

```
    [subject=s, credit=c,
     [right1=value1,right2=value2,...]]
```

Here, the binding `credit` corresponds to $ATT(s)$; `s` is the identity of the subject; `c` is a constant indicating the credit of the subject; and `[right1= value1, ...]` corresponds to $ATT(o, r)$ and represents the list of allowable rights on the $object$ and the corresponding value.

Finally we use the rewrite rule $allowed(\cdot)$ to express the UCON$_{\mathrm{ABC}}$ predicate $allowed(\cdot)$ as follows:

$$\begin{aligned} allowed(S, O, R) \quad : \quad & \mathrm{license}(O, Cs, Bs_1) \rightarrow \mathrm{license}(O, Cs, Bs_2) \\ \Leftarrow \quad & Cs \vdash canallow(S, R, Bs_1, Bs_2) \end{aligned}$$

Here, S, O, and R refer to subject, object and right, respectively; Cs is the list of clauses; Bs_1 and Bs_2 are the bindings, before and after the rule is applied.

We use LicenseScript to express explicitly the details of implementation of the UCON_{ABC} models, i.e., the authentication (`Subject==S`), condition (`M >= V`), and update (`NewM = M - V`). The explicitness gives us an advantage when implementing the model in a real system. In short, LicenseScript can bridge the gap between an abstract design and an implementation of UCON_{ABC}.

We will show how we use LicenseScript to model rights control in more detail in part II (pp.29).

1.4 Complexity

In this section, we analyze the complexity of LicenseScript corresponding to state-of-the-art security models. To analyze the complexity of access control system, security models are used to aid in formulating the rules, which specify under which conditions, what operation is allowed; and explaining how the system performs authorization decisions [McLean, 1994].

Harrison et al. [1974] formalize a simple safety analysis, i.e., determine whether an access control system can reach a state in which an unsafe access is allowed. A subject should be able to tell whether what she is about to do, e.g. give away a right can lead to the further *leakage* of that right to some unauthorized subjects. They show that any Turing machine can be simulated by an access control system. Harrison et al. [1974] define six primitive operations, such as "enter a right in a cell of an access control matrix". They also define a command to encapsulate a list of conditions, in which a sequence of primitive operations can be executed. Harrison et al. [1974] prove three complexity results of the safety problem for the HRU model:

1. Safety is undecidable in general.

2. Safety is decidable for a mono-operational system (**NP**-complete).

3. Safety is decidable within the **PSPACE**-complete complexity class with the restriction that no subjects or objects are allowed to be created [Li and Tripunitara, 2004].

As stated in 1, in general safety in the HRU model is undecidable. However, as stated in 2, HRU does have decidable safety property for the mono-operational case, where each command only allows a single primitive operation.

It is difficult to encode the LicenseScript model into the HRU model and to use the existing results for several reasons: (1) The request decision, which is based on the attributes of the subject, object and right in LicenseScript would not be mono-conditional in the resulting HRU command [Sandhu, 1992]; (2) The alteration on the attributes, e.g. to create a new subject for delegation, is unlikely to be mono-operational in HRU command [Li and Tripunitara, 2004].

Therefore, safety analysis is more complex in LicenseScript than in HRU. It is clear that the safety problem (i.e., deciding whether – given a multiset and a set of rules – it will eventually be possible to fire a given rule with given arguments) is undecidable. In fact, it is not even possible to restrict the system to one in which the safety problem is decidable without losing one of the essential features of LicenseScript: that one should be able to acquire new licenses. In fact, acquiring new licenses usually allows for firing new rules.

While safety is in general undecidable, it is useful to see if it is possible to isolate situations in which this is not the case. Here we want to point out that there exist at least two techniques that can be used to prove the safety of a given configuration. The first is by using *invariants*, the second is by showing that the system is *terminating* and *finitely branching*. In this second case, it is possible to carry out a full search of the multisets space generated by the (repeated) application of the rules and thus to establish whether a given unsafe configuration is reachable or not. We now want to spend a couple of words showing how this can be done in practice, by borrowing some techniques from the study on the termination of logic programs (i.e., the foundation of LP). In the light of what we said in the previous paragraph, we have to require a number of restrictions:

1. Rules have ground (variable-free) heads. If this condition is not sat-

isfied we immediately obtain a system that is not finitely branching. In some cases (i.e., if rules are called with arguments taken from a finite set) this condition can be met by instantiating the rules' arguments in all possible ways. Another way of avoiding this restriction is by applying abstract interpretation to the head's positions that might take an infinite number of arguments.

2. The starting mustiset configuration consists of atoms of the form $i(o, c, b)$, where o is ground, c is a set of clauses, and b is a set of bindings whose right-hand side consists of ground terms. This condition is always met in practice.

3. Rules have the form:

$$\begin{array}{rcl} r & : & a_1(o_1, c_1, b_1), \ldots, a_m(o_m, c_m, b_m) \rightarrow \\ & & i'_1(o'_1, c'_1, b'_1), \ldots, i'_n(o'_n, c'_n, b'_n) \rightarrow \\ & \Leftarrow & c_1 \vdash p_r(b_1, \ldots, b_m, b'_1, \ldots, b'_n) \end{array}$$

   Where (1) for each i, o_i and o'_i are ground, while c_i, c'_i, b_i and b'_i are variables. In addition, (2) each c'_i is equal to some c_j.

   The condition (1) is always met in practical deployments of the system and simplifies the discussion, while (2) is needed to make sure that the programs carried by the licenses are not modified at runtime.

4. Given: any rule r (notation as in point 3.), any atom $a(o, c, b)$ in the initial multiset, any set of ground terms $t_1, \ldots, t_m$, and distinct variables $x_1, \ldots, x_n$, we require that the query $p_r(t_1, \ldots, t_m, x_1, \ldots, x_n)$ is terminating and finitely branching in c.

   This is needed to ensure that the decision process terminates and that the whole system remains finitely branching. Notice that in pure logic programming a terminating system is always finitely branching (in which case one can use standard LP techniques [Apt and Pedreschi, 1993; Bossi et al., 1991; De Schreye and Decorte, 1994; Etalle and Gabbrieli, 1999]) but that this is not necessarily the case if the program can interact with the outside world.

5. There exists a level mapping $|\ |\ :\ \mathcal{B} \rightarrow \mathcal{W}$, where $\mathcal{B}$ is the set of all possible binding sets, and $\mathcal{W}$ is a set with a well-founded ordering $<$, such that given any rule r (notation as in point 3.), any atom $a(o, c, b)$ in the initial multiset, and $b_1, \ldots, b_m, b'_1, \ldots, b'_n \in \mathcal{B}$, if $p_r(b_1, \ldots, b_m, b'_1, \ldots, b'_n)$ succeeds in c then

   $$|b_1|, \ldots, |b_m| \succ_m |b'_1|, \ldots, |b'_n|$$

   Where $\succ_m$ is the (well-founded) multiset ordering induced by the ordering $>$ on $\mathcal{W}$[1]

We can finally state the result we are aiming at.

Proposition 7. *If the initial multiset and the set of rules satisfy the above conditions* $1, \ldots, 5$*, then the system is terminating and finitely branching, and therefore safety is decidable.*

Proof (sketch). Each action of the system requires (a) selecting a rule, (b) selecting a multiset m_1 (the lhs of the rule), (c) firing a condition in a program c (contained in an atom of m_1), (d) rewriting m_1 with m_2. Action (a) is terminating and finitely branching thanks to the fact that by condition 1 rules have no nonground arguments. Action (b) is clearly terminating (the multiset is finite). Notice that condition 3 guarantees that, no matter how the multiset is being rewritten, the set of programs (set of clauses in the licenses) we have to deal with does not change; therefore, (c) terminates because of condition 4. Finally, condition 5 guarantees that the new multiset is smaller than the original one according to a well-founded ordering. This implies that it is not possible to have an infinite chain of actions. QED.

[1] Quoting from [Apt, 1997]: The *multiset ordering* is an ordering on finite multisets of natural numbers. It is defined as the transitive closure of the relation in which X is smaller than Y if X can be obtained from Y by replacing an element a of Y by a finite (possibly empty) multiset of natural numbers each of which is smaller than a. In symbols, first we define the relation $\prec$ by: $X \prec Y$ iff $X = Y - \{a\} \cup Z$ for some $a \in Y$ and Z such that $b < a$ for $b \in Z$, where X, Y and Z are finite multisets of natural numbers and then define the multiset ordering $\prec_m$ as the transitive closure of the relation $\prec$.

1.5 Summary

Digital content has become extremely common due to the rapid development of the Internet and digital technologies. Illegal use of digital content has caused huge loss to the content industry. Furthermore, many experimental, novel, and complex usage scenarios of digital content have emerged. Therefore, rights control over digital content is mandatory.

Access control is limited in its ability to control rights on digital content. To overcome some of the limitations, usage control is proposed by Sandhu and Park. However, usage control does not consider the mutability of rights. This is useful to model administrative rights, delegation rights and reverse rights. Therefore, we propose rights control as a further extension of usage control.

We will provide additional background on two main aspects of rights control, namely rights expression languages and rights enforcement mechanisms later in part II (pp.29) and part III (pp.95), respectively.

Part II
RIGHTS EXPRESSION LANGUAGES

A rights expression language (REL) is a domain specific language designed to enable efficient processing by machines of rights on digital content. RELs for instance, can be used to describe an agreement between a content provider and a music distributor, or to express the copyright associated with a given piece of music, or by specifying under which condition the user is allowed to play, broadcast or copy the music.

In chapter 1, we have briefly explained the models of access control, usage control and rights control. Here, we describe some languages that are in use to specify access control, usage control and rights control.

Access Control Matrix

An access control matrix (ACM) [Gollmann, 1999] states the rights of each subject on each object in a form of table. Table 1.3 shows an example of an ACM, which for instance, states that Alice and Charles possess the right to read (r) and write (w) File X, whereas Bob is only allowed to read File X.

User	File X	File Y	File Z
Alice	r,w	r	r
Bob	r	r,w	-
Charles	r,w	-	-

Table 1.3: An example of an access control matrix.

We can implement an access control matrix in two manners: rights can be kept with the subjects or with the objects.

In the first case, a subject is given a *capability*, which is this subject's access right. This capability corresponds to the subject's row in the access control matrix. For example, the access rights of Table 1.3 given as capabilities are shown in Table 1.4.

Alice's capabilities	–	File X: r,w; File Y: r; File Z: r
Bob's capabilities	–	File X: r; File Y: r,w
Charles's capabilities	–	File X: r,w

Table 1.4: Capabilities of subjects.

Typically, capabilities are associated with discretionary access control (as discussed in chapter 1). With capabilities it is difficult to get an overview of who has permission to access a given object. In addition, it is difficult to revoke a capability when there is a huge population of subjects. However, capability-based access control can be used in distributed systems where the access control has to deal with users roaming within a computer network.

An access control list (ACL) stores the access rights to an object with the object itself. An ACL therefore corresponds to a column of the access control matrix and states who may access a given object. The access rights of Table 1.3 given in the form of ACLs are shown in Table 1.5.

ACL for File X	–	Alice: r,w; Bob: r; Charles: r,w
ACL for File Y	–	Alice: r; Bob: r,w; Charles: r
ACL for File Z	–	Alice: r

Table 1.5: ACLs of object.

ACLs are simple to implement in relatively small systems with a relatively constant user population, but ACLs are inefficient for large-scale systems, e.g. a corporate system. It is time-consuming to check a large ACL to decide the rights of the subject, or to remove a subject from all(!) ACLs.

We can encode an access control matrix by using a first-order logic [see Abadi, 2003]. For instance, we can encode Alice's capabilities shown in Table 1.4, as shown in Figure 1.4.

In Figure 1.4, we use predicate `alice(X,Y)` to indicate that `alice` has access right Y on content `X`. We can also use predicate `bob(X,Y)` to express Bob's capabilities. Similarly, we can use first-order logic to encode the access control list of File X (Table 1.5) in Figure 1.5. The predicate `fileX(X,Y)` denotes that the subject `X` has access right `Y` on

fileX.

```
alice(fileX,r)
alice(fileX,w)
alice(fileY,r)
alice(fileZ,r)
```

Figure 1.4: Logic representation of Alice's capabilities shown in Table 1.4.

```
filex(alice,r)
filex(alice,w)
filex(bob,r)
filex(charles,r)
filex(charles,w)
```

Figure 1.5: Logic representation of access control list of File X shown in Table 1.5.

An access control matrix is only capable of specifying access control. It is limited in its capability to express obligation and condition. Therefore, it cannot satisfy requirements of modern applications.

Several more advanced languages based on first-order logic have been proposed to meet different requirements of various application domains shown in Figure 1.6. We explore these languages in the following sections.

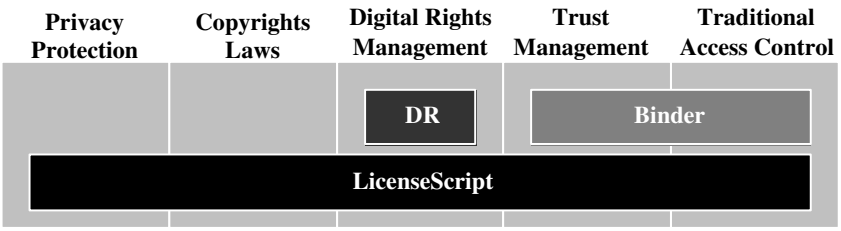

Figure 1.6: Application domain covered by access control language (light gray box), usage control language (dark gray box) and rights control language (dark box).

Access Control Language

DeTreville's Binder [DeTreville, 2002] is a logic-based access control language. Binder is based on Datalog, which is a subset of the well-known Prolog logic-programming language [Lloyd, 1987].

```
can(alice,r,fileX).
can(alice,w,fileX).
can(bob,r,fileX).
can(charles,r,fileX).
can(charles,w,fileX).
can(alice,r,fileY).
can(bob,r,fileY).
can(bob,w,fileY).
can(alice,r,fileZ).
```

Figure 1.7: The Binder encoding of the access control matrix Table 1.3.

We can encode the access control matrix of Table 1.3 by using Binder as shown in Figure 1.7. The Binder predicate `can(X,Y,Z)` denotes that subject `X` has an access right `Y` on content `Z`.

Binder can express more advanced access control mechanisms. For instance, Binder can express mandatory access control and role-based access control (as discussed in chapter 1) by extending the predicate `can` as shown in Example 8.

Example 8. Employee of BigCo can read File X.

```
can(X, r, fileX) :-
    employee(X, bigco).
```

In Example 8, predicate `employee(X, bigco)` indicates if subject `X` is an employee, i.e., role, of the company `bigco`. Example 8 is a Binder program specifying that `X` must be an employee of `bigco` so that she can read (`r`) `fileX`.

Binder can encode more than an access control matrix, e.g. delegation and trust, by using application-specific predicates [DeTreville, 2002]. Therefore, as shown in Figure 1.6, Binder can also express trust management in addition to traditional access control.

Usage Control Language

Binder uses attributes of a subject and an object to decide if an access right can be granted. For instance, in Example 8, `employee` is an attribute

(role) of a subject.

Usage control, as discussed in chapter 1, introduces two additional decision factors in deciding a right on an object to a subject, namely obligation and condition. An example of a language that offers these features is Gunter et al. [2001] logic-based language "Digital Rights" (DR). To illustrate their language, we use Example 9.

Example 9. The content can be rendered every time after the subject pays.

$$\begin{aligned}&\text{license}(t_0, x, p, w, d) \ = \\ &\quad \{t_0 : \mathbf{pay}[x], t_1 : \mathbf{render}[w, d] \mid t_0 < t_1 < t_0 + p\}\end{aligned}$$

As shown in Example 9, two types of actions are defined in DR, namely $\mathbf{render}[w, d]$ and $\mathbf{pay}[x]$. Action $\mathbf{render}[w, d]$ represents rendering of content w by device d; and action $\mathbf{pay}[x]$ represents making a payment of amount x.

An action (a) is always paired with a time t, i.e., $t : a$, which indicates that at time t an action a is performed. Therefore, the license of Example 9 denotes that an amount of money x must be paid at time t_0 so that the content w can be rendered by device d for a period of time p.

The action $\mathbf{pay}[x]$ represents an obligation; t represents a time-based condition; and d (in action $\mathbf{render}[w, d]$) is a location/system-based condition. In short, their logic is able to express some obligations and conditions.

DR is a high-level abstract language. It abstracts usage actions into two genres, i.e., $\mathbf{render}$ and $\mathbf{pay}$. It is not able to distinguish different types of actions. Therefore, it is not able to encode access rights stated in an access control matrix of Table 1.3 explicitly.

In addition, DR is too abstract to model more complex usage scenarios, for example DR cannot model a scenario where a user is required to register as a member before renting a video.

Rights Control Language

As discussed in chapter 1, the rights control model associates mutable attributes with a right. This allows us to model more sophisticated rights, e.g. derivative right, administrative right, and delegation right.

We have briefly introduced LicenseScript [Chong et al., 2003a] in chapter 1. In this part of the thesis, we introduce the design rationale of LicenseScript in chapter 2.

Chapter 3 explains LicenseScript further by using a number of usage scenarios, which cover a wide variety of application domains, as shown in Figure 1.6. We also compare prominent XML-based rights expression languages, i.e., XrML and ODRL, with LicenseScript. Chapter 4 explores another application domain of LicenseScript: to model a difficult aspect of copyright law, i.e., Fair Use.

CHAPTER 2

LICENSESCRIPT

LicenseScript: A Novel Digital Rights Language and its Semantics[1]

Cheun Ngen Chong, Ricardo Corin, Sandro Etalle, Pieter Hartel, Willem Jonker, and Yee Wei Law

Abstract We propose LicenseScript as a new multiset rewriting/logic-based language for expressing dynamic conditions of use of digital assets such as music, video or private data. LicenseScript differs from other rights expression languages in that it caters for the intentional but legal manipulation of data. We believe this feature is the answer to providing the flexibility needed to support emerging usage paradigms of digital data. We provide the language with a simple semantics based on traces.

[1]This chapter has been published in 3rd International Conference on Web Delivering of Music (WEDELMUSIC), 2003, pages 122–129, IEEE Computer Society Press.

2.1 Introduction

Most information, such as books, music, video, personal data and sensor readings (we generalize this information as *data*), is intended for a specific use. This specific use should conform to particular terms and conditions, which are often governed by *licenses*. To describe a license, a specific language is needed. In fact, the last few years have witnessed a proliferation of *rights expression language* (REL) (also known as digital rights language (DRL)). These are usually based on XML, e.g. XrML [Guo, 2001] (`http://www.xrml.org`) and ODRL [Iannella, 2001] (`http://www.odrl.net`).

It is now widely acknowledged that the above-mentioned XML-based RELs have some important shortcomings: (1) the syntax is complicated and obscure when the conditions of use become complex, (2) these languages lack a formal semantics [Gunter et al., 2001; Pucella and Weissman, 2002]; the meaning of licenses relies heavily on the human interpretation, and (3) the language cannot express many useful copyright acts [Mulligan and Burstein, 2002]. Gunter et al. [2001] overcome some of the drawbacks by introducing an abstract model and language with a corresponding formal semantics. Pucella and Weissman [2002] follow up Gunter et al's effort with more rigor.

Licenses play an important role in the electronic distribution of music. With the advent of the Internet, music labels are searching for ways of distributing music over the Internet in a way that respects the rights of the owners and the labels. At the same time, partly due to Napster, users have become used to easy access to music on their devices. As a result, *Electronic Music Distribution* (EMD) will only be successful if these systems provide flexible support for licensing of music and the user gains a broad freedom in accessing the music, which in turn requires flexible and easy to understand licenses will a well defined semantics. In an attempt to cope with the requirement of seamless music access on users devices, the notion of authorized domain [van den Heuvel et al., 2002] has been developed (`http://www.dvb.org`).

An authorized domain can be seen as the collection of devices that belongs to a user or a household. The idea is that music is delivered to the authorized domain, and that it can be accessed seamlessly on any device

in that domain. The music access is governed by licenses that are bound to the domain, rather than to individual devices. In addition to licenses that govern the access within the domain, there are licenses that govern the exchange of music between domains. The latter guarantees that unauthorized music exchange can be prohibited. Regrettably, state-of-the-art languages cannot cope with this scenario.

In this paper, we propose LicenseScript, a language that is able to express conditions of use of dynamic and evolving data in authorized domains. LicenseScript is based on (1) multiset rewriting, which is able to capture the *dynamic* evolution of licenses, (2) logic programming, which captures the static terms and conditions on a license, and (3) a judicious choice of the interfacing mechanism between the static and dynamic domains. LicenseScript makes it possible to express a multitude of sophisticated usage patterns precisely and clearly. The formal basis of LicenseScript (multiset rewriting and logic programming) provides for a concise and explicit formal semantics.

We use Prolog program for the license clauses because of the double declarative and procedural reading of Prolog. Thanks to its declarative reading, Prolog is suitable for rendering licenses, which can easily be read and understood by humans. In fact, Prolog is often used as a language to describe policies and business rules. On the other hand, the procedural reading of Prolog allows for an direct execution of the code.

The organization of the remainder of the paper is as follows: Section 2.2 explains the LicenseScript language. Section 2.3 demonstrates examples for Electronic Music Distribution. Section 2.4 gives some related work. Finally, Section 2.5 gives the conclusions.

2.2 Language

In this section we describe the LicenseScript language. We start by introducing some basic concepts that are needed in the sequel.

2.2.1 Preliminaries

As mentioned earlier, LicenseScript is based on multiset rewriting. By a *multiset* (also known as a *bag*) we mean a set with possibly repeated elements; denoted with *brackets*. For example, $[a, b, b, c]$ is a multiset.

In our approach, licenses are bound to terms that reside in multisets. For the specification of these licenses, we use logic programming; the reader is thus assumed to be familiar with the terminology and the basic results of the semantics of logic programs [Lloyd, 1987]. We also use Prolog notation: we use words that start with uppercase (X, Y, $\cdots$) to denote variables, and lowercase ($music\_piece$, $video\_track$,$expires$, $\cdots$) to denote constants. We work with *queries*, that is sequences of atoms. Further, given a syntactic construct E (so for example, a term, an atom or a set of equations) we denote by *Var*(E) the set of the variables appearing in E. Given a substitution $\sigma = \{x_1/t_1, ..., x_n/t_n\}$ we denote by *Dom*(σ) the set of variables $\{x_1, \ldots, x_n\}$. A substitution σ is called a *matching substitution* of terms t and s if $t\sigma = s$, and *Dom*(σ) = *Var*(t). In that case, we say that t *matches* s. If a term matches with another one, then it follows that there exists a unique matching substitution.

We also borrow the concept of *SLD-resolution* [Lloyd, 1987] from logic programming:

Definition 10. Given a program P, and a query (i.e., a conjunction of atoms) Q, we write $P \vdash_{SLD} Q$ (or simply $P \vdash Q$) when there is a successful SLD-derivation for query Q in program P. A successful execution of a query may result in a (computed answer) substitution. $P \vdash Q$ basically means that execution of the query Q in the program P yields *success*.

2.2.2 Licenses

A license defines the terms and conditions of use for music, videos etc. Therefore, a license contains at least two relevant items of information: (i) a reference to the *data* that is being licensed, and (ii) the *conditions of use* on that data.

In our formalism, a license is represented by a term of the form $lic($ $content$, Δ, $B)$ (as can be seen in Figure 2.1) where:

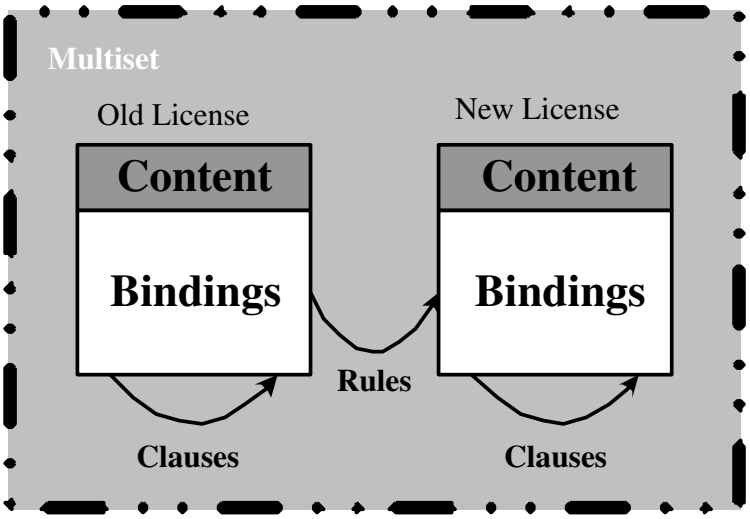

Figure 2.1: The transformation of licenses with *content* and *bindings* in a multiset caused by rules.

- *content* is a unique identifier representing the data the license refers to.

- Δ is a set of *clauses*, i.e., a Prolog program. This program defines when certain operations (like *play*) are allowed.

- B is a set of *bindings*, i.e., a set containing elements of the form $name \equiv value$. For instance $\{expires \equiv 10/10/2003\}$ is a set of bindings with just one element.

Bindings are needed to have a flexible way of storing modifiable data. A license could be regarded as a database in which Δ is the intensional part, while B is the extensional part.

To interface licenses with the external world, we have to define an *interface*, i.e., a set of reserved calls that form the "API" of the license. The precise definition of this interface is beyond the scope of this paper. For example, we use *canplay(·)* to indicate when a license allows a given piece of music to be played: if the query $canplay(B, B')$ succeeds in the program Δ, this means that the license $lic(a, \Delta, B)$ allows the piece a to

be played. Notice that we passed the set of bindings B as an argument to the query.

Example 11.

1. The following license allows to play mus:

$$lic(mus, \{canplay(X, X) : -true.\}, \{\})$$

2. The license $lic(mus, \{\}, \{\})$ does not allow any operation on mus.

3. The license

$$lic(a, \Delta, \{expires \equiv 10/10/2003\})$$

where Δ is

$$\{canplay(B, B) : -today(D),\\ get\_value(B, expires, Exp), Exp > D.\}$$

allows to play *a* until the given expiration day.

$today(D)$ and $get\_value(B, n, V)$ are two primitives that work as follows: $today(D)$ binds the variable D to the current system date, while $get\_value(\ B,\ n,\ V)$ reports in V the value of the name n according to the set of bindings B. In the remainder, we gather all such primitives in a special program that we call the *domain*, denoted D. Notice that there can be many domains in which licenses reside, and probably a domain will have different meanings for the primitives than another domain.

There are situations in which the "execution" of a license should be followed by the creation of a new set of bindings for the next step in the evolution of the license. Consider for instance a license that allows to play a piece of music only a given number of times: every time a *play* action is carried out, a counter should be incremented. This is done by means of the primitive $set\_value(OldB, name, value, NewB)$. This primitive allows a name from a binding to be associated with a new value, which we use to support the evolution of licences. Consider, for instance, the

following license: $lic(a, \Delta, \{played\_times \equiv 3\})$, where Δ consists of the following clause:

$$\begin{aligned} canplay(B, B') \ :- \\ \quad & get\_value(B, played\_times, R), R < 10, \\ \quad & set\_value(B, played\_times, R + 1, B'). \end{aligned}$$

Here, we first extract the value of variable $played\_times$ into local variable R. Then, if we have not exhausted the possible playing times allowed by the license (in this case, 10), we proceed to increase the value of $played\_times$ from bindings B to $R + 1$, into new *output* bindings set B'.

2.2.3 The Rules

Licenses typically reside inside a device. The modelling of communication between devices and the licenses is done by means of *rewrite rules*. These rules can be thought as the *firmware* of the device; licenses may come and go from a device, but the rules are fixed into the device (however, rules can be 'updated' once in a while). The syntax of rules we adopt is that of *multiset rewriting* (we use, in particular, Gamma notation [Banâtre et al., 2001]).

Definition 12. A rewrite rule is a 4-tuple

$$rule(arg) : lms \rightarrow rms \Leftarrow cond$$

where $rule(arg)$ is an atom called *rule label*, lms and rms are two multisets, and $cond$ is a sequence of elements of the form $P_i \vdash Q_i$.

Notice that a substitution σ can be applied in a natural way to a rule: $\sigma(\ rule(arg) : lms \rightarrow rms \Leftarrow cond) = rule(\sigma(arg)) : \sigma(lms) \rightarrow \sigma(rms) \Leftarrow \sigma(cond)$.

Intuitively, rules can be applied to a "target" multiset *MS*, if the following two conditions hold:

- First, the left multiset lms has to match some sub-multiset of *MS*; this sub-multiset is to be replaced with (right) multiset rms;

- Second, the conditions $cond$ of the rule must hold; This is done by executing all the queries in $cond$, and checking that the result is *success*.

We formalize the meaning of rule execution in the next section. An example for a rule is the following:

$$\begin{array}{rcl} play(X) & : & lic(X, \Delta, B) \rightarrow lic(X, \Delta, B') \\ & \Leftarrow & \Delta \vdash canplay(B, B') \end{array}$$

This rule can be applied to a license $lic(X, \Delta, B)$, replacing it with another license $lic(X, \Delta, B')$ if condition $\Delta \vdash canplay(B, B')$ holds.

2.2.4 LicenseScript Execution Model

As we already mentioned, licenses are represented by terms of the form $lic(\ content, \Delta, B)$. For the sake of exposure, we assume that all available licenses are stored in a given multiset *MS*.

Definition 13. Rule execution. Given two multisets *MS* and *MS'*, a rule $label : l \rightarrow r \Leftarrow cond$, and an atom a (called the request action), we write *MS* $\xrightarrow{a\sigma}$ *MS'* if:

1. a matches with $label$, with matching σ_1.
2. $l\sigma_1$ matches with T', with matching σ_2, for some sub multiset T' of *MS*.
3. For each $\Delta_i \vdash \phi_i$, $i \in [1, n]$ in $cond$, $(\Delta_i \vdash \phi_i)\sigma_1\sigma_2\delta_1 \cdots \delta_{i-1}$ succeeds, with computed answer substitution δ_i;
4. *MS'* is the result of removing each term in T' from *MS*, and appending $r\sigma_1\sigma_2\delta_1 \cdots \delta_n$ to it, then, σ is the composition of σ_1, σ_2, $\cdots$, i.e. $\sigma_1 \circ \sigma_2 \circ \cdots$.

Step 1 of this definition represents the choice of a rule for executing a given request action a (e.g., $play(mus)$) from the environment. Notice that there may be more than one rule that matches with the request action, so a first source of non-determinism appears here. The request action

can also *fail* if no rule matches with the request. After a rule is chosen, *Step 2* finds a set of licenses T' in the multiset *MS* to which the rule can be applied. Again, different choices of T' may produce non-determinism here. This corresponds to the possible situation in which the user possesses more than one license (or set of licenses) that allows her to effectuate the desired action. In this case, we can assume that the system asks the user which license should be used. *Step 3* checks that the conditions of the rule hold, by executing the queries with the carried out substitutions. Finally, *Step 4* transforms the multiset by applying the replacement specified by the rule.

Example 14. Consider the multiset containing the following licenses: *MS* $= [lic(music, \Gamma, C), lic(video, \Sigma, D)]$, where $C = \{played\_times \equiv 2\}$, $D = \{played\_times \equiv 10\}$, and $\Gamma = \Sigma =$

$$\begin{aligned}\{&canplay(B, B') :- \\ &\quad get(B, played\_times, N), N < 10, \\ &\quad set(B, played\_times, N + 1, B')\}\end{aligned}$$

Let R be the singleton rule set containing the following rule:

$$\begin{aligned} play(X) \quad &: \quad lic(X, \Delta, B) \rightarrow lic(X, \Delta, B') \\ &\Leftarrow \quad \Delta \vdash canplay(B, B') \end{aligned}$$

1. Suppose the environment requests the action $play(music)$. This will match rule $play(X)$, giving matching $\sigma_1 = \{X/music\}$.

2. The next step involves looking for occurrences of $lic(music, \Delta, B)$ in *MS*. The only possible match is $lic(music, \Gamma, C)$. This gives us matching $\sigma_2 = \{\Delta/\Gamma, B/C\}$.

3. Condition $\Gamma \vdash canplay(C, B')$ has to be evaluated. Since variable $played\_times$ is less than 10 in C, the $canplay(C, B')$ succeeds in the Prolog program Γ, hence the condition is satisfied. We get the computed answer substitution $\delta_1 = \{B'/played\_times \equiv 3\}$.

4. Finally, *MS* is updated. License $lic(music, \Gamma, C)$ is removed from *MS*, and replaced by $lic(music, \Gamma, C')$, where $C' = \{played\_times \equiv 3\}$.

Example 15. Consider the same multiset and rules of the previous example. Suppose now request action $play(video)$ is issued. This action, even though has a matching rule and a matching license in the multiset, cannot be carried out completely. This is so since, in the unique matched license (that is, $lic(video, \Sigma, D)$) condition $\Delta \vdash canplay(B, B')$ does not hold, the video has been played too often.

Definition 13 describes how a multiset can evolve to another by means of executing a rule. The precise notion of multiset execution, which can be understood as the semantics of LicenseScript, can be then described as sequences of rule execution.

Definition 16. Given a multiset MS and a set of rules R, an *execution* is the (possibly infinite) sequence of rule applications $MS \xrightarrow{a_1\sigma_1} MS_1 \xrightarrow{a_2\sigma_2} MS_2 \cdots$. The *trace execution* of MS is defined as $a_1\sigma_1 \cdot a_2\sigma_2 \cdots$.

The semantics of executing a multiset and a rule set is then defined as all possible trace executions, according to the above definitions.

2.3 Electronic Music Distribution

In this section we provide examples in Electronic Music Distribution, to show the flexibility of LicenseScript.

2.3.1 Authorized Domains

As explained in the introduction, an authorized domain can be seen as the collection of devices belonging to a household. In this paper we only focus on the relationship between licenses and ADs, and we assume the existence of an AD implementation that deals with domain and content management issues [van den Heuvel et al., 2002], i.e. we assume the presence of compliant devices that are governed by AD management rules. We first show how a license can be bound to a specific domain. Consider the following license:

$$lic(mus, \Delta, \{in\_domain \equiv cert\})$$

where $\Delta =$

$$\begin{aligned}\{validD(B) \;\; :- \;\; & get\_value(B, in\_domain, Id_2),\\ & identify(Id_1), Id_1 = Id_2.\}\end{aligned}$$

Here, $identify(Id_1)$ is a primitive which is used to retrieve the identity of the current domain. Clause $validD(\cdot)$ checks that the current domain is in fact the domain to which the license is bound. A rule like $play(\cdot)$, for instance, can now be defined as only valid if the license is in the allowed domain:

$$\begin{aligned}play(Mus) \;\; : \;\; & lic(Mus, \Delta, B) \rightarrow lic(Mus, \Delta, B)\\ \Leftarrow \;\; & \Delta \vdash validD(B)\end{aligned}$$

Now we are ready to illustrate a slightly more complex example: Consider a license that allows a certain piece of music to be played only for a limited time within the domain. This license can be written as follows:

$$\begin{aligned}&lic(mus, \Delta,\\ &\quad \{in\_domain \equiv cert, expires \equiv expiration\_date\})\end{aligned}$$

where Δ consists of the following two clauses:

$$\begin{aligned}validD(B) \;\; :- \;\; & get\_value(B, in\_domain, Id_1),\\ & identify(Id_1), Id_1 = Id_2.\\ nexpired(B) \;\; :- \;\; & get\_value(B, expires, Exp),\\ & today(D), D \leq Exp.\end{aligned}$$

Finally, the corresponding *play(·)* rule can be defined:

$$\begin{aligned}play(Mus) \;\; : \;\; & lic(Mus, \Delta, B) \rightarrow play(Mus, \Delta, B)\\ \Leftarrow \;\; & \Delta \vdash validD(B), \Delta \vdash nexpired(B)\end{aligned}$$

2.3.2 Payment

To address the different forms of payment, we first model a *wallet*. Then, we show how various payment methods are implemented in LicenseScript.

2.3.2.1 Modelling a Wallet

We model the existence of a *wallet*, in our approach, as a another element of the multiset. The wallet is represented as a term of the form *wallet*(Δ, B), where Δ is a Prolog program, and B are a set of bindings. Similarly to licenses, in the wallet we have clauses that allow rules to perform operations over the wallet. We assume that one binding named m, which represents the amount of money in the wallet, is always in B.

A clause that may reside in Δ is *canload*$(\cdot)$, which is used to load or increase the balance of the wallet, as can be shown as follows:

$$\begin{aligned} &canload(A, B, B') \ :- \\ &\quad get\_value(B, m, M), set\_value(B, m, M + A, B'). \end{aligned}$$

where A is the amount of money the user likes to load onto the wallet.

Using clause $canload(\cdot)$, a rule that loads money into the wallet can now be written:

$$\begin{aligned} load(Amount) \quad &: \quad wallet(\Delta, B) \rightarrow wallet(\Delta, B') \\ &\Leftarrow \quad \Delta \vdash canload(\textbf{\textit{Amount}}, B, B') \end{aligned}$$

Another useful clause in the wallet is $cantransfer(\cdot)$, used to transfer a certain amount of money to another entity (e.g., a content provider):

$$\begin{aligned} &cantransfer(P, A, B, B') \ :- \\ &\quad get\_value(B, m, M), A \leq M, \\ &\quad set\_value(B, m, M - A, B'), transfer(P, A). \end{aligned}$$

where primitive $transfer(P, A)$ models the money transfer to entity P of the amount of money A.

2.3.2.2 Payment Methods

There are at least three common alternatives paymen models, as described in [Gunter et al., 2001]: (1) pay *per*-use, a payment is issued *each time* the content is used; (2) pay *upfront*, the content can be used after the payment has taken place, for a period of time p; and (3) pay *flatrate*: The content is

used, and then the payment must be made at the end of the content usage We now illustrate the implementation of the pay per-use and pay upfront methods in LicenseScript. We leave aside pay flatrate, as it is similar to pay per-use.

Pay Per-Use We can model pay per-use in LicenseScript by means of including in a license the following clause $canplay(\cdot)$:

$$\begin{aligned} &canplay(P, A, B) \ :- \\ &\quad get\_value(B, provider, P), \\ &\quad get\_value(B, amount, A). \end{aligned}$$

where $provider$ is the binding representing the content provider, while binding $amount$ represents the cost to play music track. Intuitively, clause $canplay$ returns the price of the content in A and the provider who should receive the payment, in variable P. This allows a rule calling this clause to perform the required payment:

$$\begin{aligned} play(Mus) \quad : \quad & lic(Mus, \Delta, B), wallet(\Gamma, C) \rightarrow \\ & \quad lic(Mus, \Delta, B), wallet(\Gamma, C') \\ \Leftarrow \quad & \Delta \vdash canplay(P, A, B), \\ & \quad \Gamma \vdash cantransfer(P, A, C, C') \end{aligned}$$

Here, clause $canplay(P, A, B)$ retrieves P and A, which clause $cantransfer$ (P, A, C, C') uses to perform the actual money transfer.

Pay Upfront For modelling this payment method, we need to use two different clauses and rules, since the actual payment and content usage may differ in time: the payment is first done, and only later the content is used.

We first define a clause $canpay(\cdot)$ for paying as follows:

$$\begin{aligned} canpay(P, A, B, B') \ :- \\ & get\_value(B, paid, Paid), Paid = false, \\ & get\_value(B, provider, P), \\ & get\_value(B, amount, A), today(D), \\ & set\_value(B, period, D + fp, B'), \\ & set\_value(B, paid, true, B'). \end{aligned}$$

Here, binding $paid$ is a flag that represents whether the content has already been paid or not. Binding $period$ is used to store the allowed period of time for playing the content, and constant fp represents the period of time in which the content can be accessed after the payment.

Clause $canpay(Provider, Amount, B, B')$ first checks that the payment has not been done yet. It then returns the value of the provider and the amount in variables $Provider$ and $Amount$. After this, the period of allowed use is set appropriately, and finally flag $paid$ is set to $true$, indicating the payment.

Using the above clause we can define the rule for pay upfront:

$$\begin{aligned} pay(Mus) \ : \ & lic(Mus, \Delta, B), wallet(\Gamma, C) \\ & \rightarrow lic(Mus, \Delta, B'), wallet(\Gamma, C') \\ & \Leftarrow \Delta \vdash canpay(Provider, Amount, B, B'), \\ & \Gamma \vdash cantransfer(Provider, Amount, C, C') \end{aligned}$$

Now, we can define the $canplay(\cdot)$ clause for action ***play***, which will allow the content to be played only if the payment has been done, and the allowed period of time has not expired:

$$\begin{aligned} canplay(B) \ :- \\ & get\_value(B, paid, Paid), Paid = true, \\ & today(D), get\_value(B, period, P), D \leq P. \end{aligned}$$

Finally, we define the rule for $play(\cdot)$:

$$\begin{aligned} play(Mus) \quad : \quad & lic(Mus, \Delta, B) \rightarrow lic(Mus, \Delta, B) \\ \Leftarrow \quad & \Delta \vdash canplay(B) \end{aligned}$$

2.3.3 Clipping Licenses

Suppose a user who has purchased a music track from a content provider requires some comments from other users. In LicenseScript, she can *clip* the license, and then she can send the clipped licenses to other people as a preview or recommendation. Notice that this operation splits the license but not the content.

The license in question looks like this:

$$lic(mus, \Delta, \{start \equiv 0, end \equiv mus\_length\})$$

where Δ contains the following $canclip(\cdot)$ clause:

$$\begin{aligned} canclip(S, E, B, B') \ :- \\ & get\_value(B, start, OS), \\ & get\_value(B, end, OE), S \geq OS, \\ & E \leq OE, set\_value(B, start, S, B'), \\ & set\_value(B, end, E, B'). \end{aligned}$$

Bindings $start$ and end are markers that indicate the head and tail of the music track.

The corresponding rule for $clip$ operation can now be defined:

$$\begin{aligned} clip(S, E, Mus) \ : lic(Mus, \Delta, B) \rightarrow \\ & lic(Mus, \Delta, B), lic(Mus, \Delta, B') \\ & \Leftarrow \Delta \vdash canclip(S, E, B, B') \end{aligned}$$

Note that the content is still the same, full-length here; only the start and end markers are modified. A different clip operation that includes the actual production of a new, clipped content, would need the use of a primitive that performs the operation.

A special case of the *clipping* operation occurs when we want to duplicate a license. This operation is often needed. In fact, one of the primary requirements of LicenseScript architecture is that devices do not have to be always on: in particular, we do not want the system to be dependent from the reacheability and the availability of *domain server*. To implement correctly the concept of authorized domain, we then have to be able

to duplicate licenses. Consider for instance the situation of a person that has the license for listening to a piece of music within his home and who rightfully wants to listen to it also while driving her car. If devices are not always on, then the car device might be incapable of checking on the home server for the presence of the right license. Therefore, there has to be a license for the music in the car device as well, and this is possible only if we can duplicate licenses. Thus, by allowing users to duplicate the licenses confined in their authorized domain without any restrictions provide a broad freedom for the users to listen to the music *whenever*, *wherever* and *however* they like.

2.4 Related Work

In this section, we briefly discuss the related work. As mentioned in section 2.1, there are several XML-based RELs proposed, the most prominent being XrML and ODRL. The crucial difference between LicenseScript and XML-based RELs, is that the former is a (Turing-complete) language, while the latter are only suitable to describe a set of constraints, the semantics of which is given by the implementation algorithm. This makes it very difficult to compare the two approaches. A sure problem with XML-based RELs is that their syntax becomes intricated when the scenarios of licensing and content usage patterns becomes complex. As a result, the license may expand drastically in size. We aim to render the LicenseScript lightweight to be fit into small devices, i.e. with limited resources (CPU, memory, etc.). Eventually, we could compile XrML and ODRL into LicenseScript (which can be thought of as an "intermediate code") to be accommodated in small devices. LicenseScript, being executable, allows to formulate complex policies in a succinct manner like: *Pay 1USD to videos.com when you view this video for the first time; each subsequent time that you view this video the cost drops by* 105*. After ten times, the video becomes free*. To the best of our knowledge, to implement such a policy in XrML one needs to define an extension of the language, and needs to provide an implementation to it.

On the other hand, we found other concepts that are easily expressed in XrML or ODRL but would require much more work in LicenseScript.

Consider, for instance, the use of *role based access control* in ODRL. This allows one to specify, for example, "any student of this university may listen to the local university radio." Here, the right to listen to the radio is assigned to the *role* "student", and not to each physical student. Then, there is no need to deal with identification or authorization mechanisms at the specification level. In LicenseScript, on the other hand, this would not be possible because there is no way to "abstract" such high-level descriptions. In this specific case, to support role based access control we would need to implement identification and authorization procedures (possibly using a PKI infrastructure) in the same license.

Gunter et al. [2001] from InterTrust Technologies Corporation and Pucella and Weissman [2002] from Cornell University present two logics for licenses. By borrowing techniques from programming semantics [Gunter, 1992], Gunter et al. develop a model and a language for describing licenses. Their logic consists of a domain of sequences of events called *realities*. An event is modelled as a pair of a time value and an action. Note that only one event is allowed at a time. A finite set of events is embodied in a reality. A license, then, is a set of realities. Most licenses consist of infinitely many realities in order to allow the use of a work at one or more of infinitely many times during some period. Using the proposed model, Gunter et al. formularize several standard license types, which they call *simple licenses*. They are the same that we have treated in this paper: simple licenses are "Up Front", "Flat Rate" and "Per Use". Simple licenses can be used as the building blocks of more complex licenses.

Pucella and Weissman [2002] follow up Gunter et al.'s effort. They propose 3 syntactic categories: (1) action expression, (2) license, and (3) formula. The action expressions are either *permitted* or *obligatory*. In other words, they reason about the licenses and the user's actions with respect to the licenses; this is done by means of a temporal *deontic* logic. This distinction is what makes their logic more accessible and complete than Gunter et al.'s. A license is an action sequence (not to be confused with an action expression). A formula designates an action sequence that is valid for a license. At most, one action per time per license can occur.

LicenseScript uses multiset rewriting which is more expressive than the denotational semantics of Gunter et al. LicenseScript is also readily subject to logical parallelism. Pucella et al.'s logic is only a starting point,

with the assumption of one client and one provider and therefore definitely does not cater for concurrency, like LicenseScript does. To state the obvious, Pucella et al. also have not yet taken into account the malleability of licenses and contents (e.g. as a result of "clipping" and "mixing"), and the concepts of authorized domains.

2.5 Conclusions and Future Work

We propose LicenseScript, a novel digital rights language based on multiset rewriting and logic programming. We present the design of the language using a scenario that represents an elaborate pattern of content use.

LicenseScript differs from other RELs in that it has an explicit static and dynamic part. The terms and conditions on content form the static part. These terms and conditions usually derive from legal, regulatory and business rules, and are therefore appropriately expressed using Prolog clauses [Sergot et al., 1986]. A license is used in a changing context and must therefore have the ability to evolve. The dynamics are represented by interpreting a license as an element of a multiset to which multiset rewrite rules are applied. These rules represent the way in which the context (devices and systems) act upon licences. The dual nature of a license (static versus dynamic) is thus represented by a two-tier structure of LicenseScript. The two levels are linked by a set of bindings that represents the current state of the evolution.

As future work, we are currently implementing the language, using an existing DRM platform [Chong et al., 2002]. Furthermore, we plan to study in detail relevant legal, regulatory and business cases to ensure that the language is convenient to use. Licenses evolution is an interesting issue, and we plan to investigate this point further. Formal verification of license properties (e.g., safety) in a given multiset and rule set, could allow us to reason more precisely about what a license is supposed to achieve and what *actually* allow.

CHAPTER 3

USAGE SCENARIOS

Comparing Logic-based and XML-based Rights Expression Languages[2]

Cheun Ngen Chong, Sandro Etalle, and Pieter Hartel

Abstract Several rights expression languages (RELs) have been proposed to describe licenses governing the terms and conditions of content access. In this field XrML and ODRL play a prominent role. Both languages are powerful yet complex. In this paper we propose a way of analysing RELs and we apply it to ODRL, XrML and to LicenseScript, a REL we propose. In addition, we test these languages against a number of example scenarios. These examples bring new insights, and shed new light on some of the limits of XrML and ODRL.

[2]This chapter has been published in Proceedings of On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops, volume 2889 of LNCS, 2003, pages 779–792, Springer-Verlag.

3.1 Introduction

Right expression languages (RELs) are languages devised specifically to express the condition of use of digital content in general, and of multimedia in particular. RELs can for instance be used to describe an agreement between a content provider and an music distributor, or to express the copyright associated to a given piece of music, by specifying under which conditions the user is allowed to play, broadcast, or copy it.

In the vast scene of multimedia delivery, two RELs in particular have attained a prominent position: XrML [Guo, 2001] and ODRL [Iannella, 2001]. The eXtensible rights Markup Language (XrML) (`http://www.xrml.org`) is proposed and maintained by ContentGuard, Inc. (`http://www.contentguard.com`), and has its roots in the Stefik's Digital Property Rights Language. XrML is adopted by Microsoft in Windows Media Player 9. The Open Digital Rights Language (ODRL) (`http://www.odrl.net`) was proposed by Iannella from IPR Systems Ltd. (`http://www.iprsystems.com`). ODRL is endorsed by the Open Mobile Alliance (OMA) (`http://www.openmobilealliance.org`).

XrML and ODRL have many similarities: syntactically they are both based on XML while structurally they both conform to the Stefik's axiomatic principles of rights modelling (`http://www.oasis-open.org/cover/DPRLmanual-XML2.html`).

XML-based RELs, however, have some intrinsic disadvantages: (1) the syntax is complicated and obscure when the conditions of use become complex, (2) these languages lack a formal semantics [Pucella and Weissman, 2002; Gunter et al., 2001], i.e. the meaning of licenses relies heavily on human interpretation, and (3) the languages cannot express many useful copyright laws [Mulligan and Burstein, 2002]. To address these problems we have proposed a new, logic-based REL, named LicenseScript [Chong et al., 2003a]. LicenseScript has a declarative as well a procedural reading (i.e., can be used as a programming language), which makes it possible to capture a multitude of sophisticated usage patterns precisely and unambiguously.

LicenseScript provides an approach to RELs which is diametrically opposite to that of XrML and ODRL: it is logic-based rather than XML-based. This makes it difficult to make an objective assessment and com-

parison of the two REL styles. Such an objective assessment is important for a clear understanding of the advantages and the limitation of XrML and ODRL. Last but not least, it must be noted that making such assessment is far from trivial, as ODRL and XrML specifications are huge and complex.

This paper aims (at least partially) at solving the aforementioned problem (i.e. complexity of XML-based RELs). Our contribution is twofold: first, we develop an anatomy of the RELs, then we apply it to ODRL, XrML and LicenseScript; secondly, we analyse in depth a number of examples, which we have coded in LicenseScript as well as in ODRL and XrML. In our opinion, these examples bring new insights, and shed new light on some of the weaknesses of XrML and ODRL.

The organization of the remainder paper: Section 3.2 discusses the anatomy of the RELs. Section 3.3 describes briefly the LicenseScript language to make the paper self-contained. Section 3.4 discuss the results of our studies on the scenarios specified in XML-based RELs. Section 3.5 describes some of the novel scenarios in LicenseScript. Lastly, section 3.6 concludes the paper and presents future work.

Note: The reader may refer to our detailed studies of XML-based RELs scenarios in our Technical Report version of this paper [Chong et al., 2003c]. The appendix in the Technical Report lists the LicenseScript code of the XML-based RELs scenarios.

3.2 Anatomy of RELs

To aid the comparison of RELs we propose an anatomy of the RELs. Based on Stefik's axiomatic principles, the XrML and ODRL specifications, and the requirements of RELs proposed by Parrott [2001], we conclude that RELs have a structure which is shown in Fig. 3.1. The figure is presented in the form of a class diagram because this exhibits the logical relations between the components. This figure provides an abstract view of a REL.

We identify four main components, namely *subject*, *object*, *operation*, and *constraint*. We explain these components in section 3.2.1. Each of these components is logically related to other components. We elaborate

these *relations* in section 3.2.2. The components and the relations established within support a wide variety of *models* of the rights management systems. We elaborate the models in section 3.2.3. Most what follows come from the existing material, which includes Parrott [2001] and XML-based RELs specification, but we explicitly indicate the additional features of a REL.

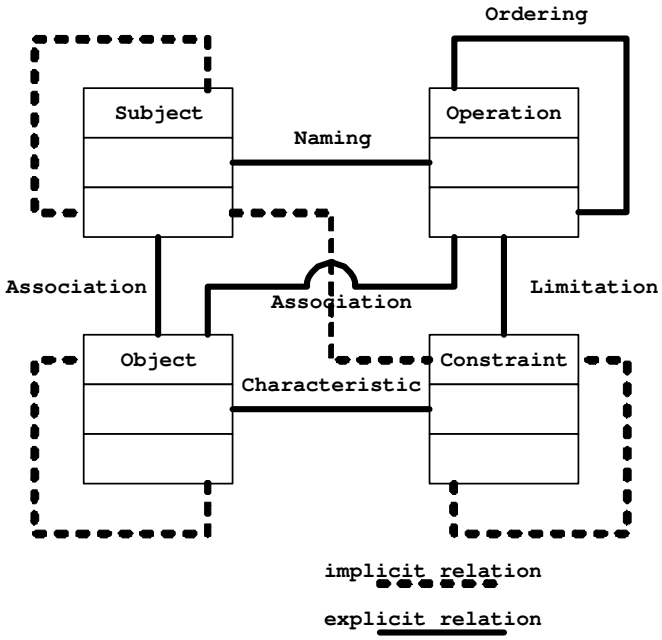

Figure 3.1: The components and their relations in a REL.

3.2.1 Components

From Parrott's requirements of RELs [Parrott, 2001] and the XML-based RELs specification, we conclude that there are four main components in a REL, namely (1) *subject*, which is an actor who performs some operations; (2) *object*, which is the content acted upon by a subject; (3) *operation*, which is what a subject can do to an object; and (4) *constraint*, which describes when an operation can be performed. There are two types of operation: a *right* is an operation that can be performed directly on the

object; and an *obligation* is an operation that must precede or follow another operation. As an illustration consider the following:

Example 17. Alice wants to play a high-quality movie three times, she must pay $5 upfront.

Intuitively, the subject is "Alice", the objects are "movie" and "$5", the right is "play", the obligation is "pay", and the constraints are "high-quality", "for three times" and "upfront".

A REL typically describes the *subject* by naming, e.g. "Alice". Additionally, the REL must be able to distinguish the type of the subject by using the names, e.g. creator, end-user, distributor and so on. A REL must also be able to specify the identification mechanism employed by common rights management systems to describe the subject, e.g. a digital certificate and public key . This is discussed in section 3.2.3.

A REL uses names to describe an *object*, such as the title of the object or the artist. A REL is required to support (1) generalized types of objects, for instance, multimedia (e.g. MP3), personal data (e.g. DOC) or meta-data (e.g. XML); (2) classification of similar objects, for instance, "publisher Addison's ebooks"; (3) fuzzy (or implicit) matching criteria of the object, for instance, "looks like" and "sounds like"; and (4) the delivery methods of the object, for example, by downloading, streaming or by means of physical storage (e.g. CD).

As mentioned earlier, there are two types of *operation*: *right* and *obligation*. There are various types of rights identified by Rosenblatt et al. [2002]: (1) *Render right*, which indicates a set of rights in which the object can be consumed, e.g. play; (2) *Reuse right*, which indicates a set of rights in which the object can be re-utilized, e.g. modify. (3) *Transport right*, which indicates a set of rights in which the subject's rights over the object can be transferred, e.g. lend. (4) *Object management*, which indicates a set of rights to handle the management over the object, e.g. move and duplicate. None of these rights cover the regulation of the rights themselves. Therefore, in addition, we propose a further set of rights, namely (5) *Rights regulation*, which indicates a set of rights that regulate the subject's rights over the object, e.g. update and renew.

A REL should be capable of describing different obligations. An obligation may be an operation that enables or activates the rights over the

objects. The *pay* and *register* operations are two common examples.

Parrott [2001] and XML-based RELs specifications recognize several common *constraints*: (1) *temporal*, such as date and time (e.g. the ebook can be viewed before 20 March 2004); accumulated (e.g. the ebook can be viewed for 2 weeks); and interval (e.g. the ebook can be viewed within 20 days from the time of issuing this license); (2) *bound*, for instance, the number of distinct times the ebook can be viewed, and the range of the page numbers of the ebook that can be printed; (3) *environment*, which may be a physical environment (e.g. geographic territory) or logical environment (e.g. network address or system environment); (4) *aspect*, which mainly relates to the technical perspectives of the object, for example, quality and format of the content; (5) *purpose*, for instance, educational purpose and commercial reason. There may be more unique constraints required when new scenarios emerge.

We introduce another constraint, namely the *status* constraint. Real time content access requires this constraint to indicate the current state, e.g. availability and accessibility of the content at the time the rights are exercised.

XrML and ODRL are able to represent render, reuse, transport, and object management rights. However, XrML and ODRL do not accommodate (explicitly) the descriptions of rights that regulate other rights. For example, "a user can *renew* the rights to play a movie within a fixed period (after the expiry time of the rights) with a discount". However, XrML and ODRL do cater for the revocation of rights and obligations. XrML does not provide explicit facilities to specify the purpose constraints. ODRL and XrML cannot express the status of the object. However, LicenseScript is able to accommodate most (if not all) the listed constraints.

3.2.2 Relations

A REL must specify *relations* between components. As can be seen in Fig. 3.1, there are two distinct types of relations, namely *explicit* relations and *implicit* relations. We use example 17 to elaborate some of the relations discussed in this section.

Parrott [2001] identifies two classes of explicit relations, namely *ordering* relation, e.g. "*pay* $5 before *play* the movie" (operation–operation);

and *association* relation, "*Alice* owns the *movie*" (subject–object) and "*play* is for *movie*" (operation–object). The *ordering* relation describes how operations are linked. For example, "pay before play" is an example of antecedent obligation; and "play then pay" is an example of consequent obligation. An ordering can be total or partial. A total ordering fully specifies the order of all operations, for example, "register, pay and then play". A partial ordering implies there is no explicit order between all items, for example, "register and then play, user can pay before or after". The *association* relation covers the subject–object and operation–object relations.

We identify three additional types of explicit relations, namely, (1) *naming* relation (subject–operation), which specifies the name of the operation the subject can perform, e.g. "*Alice plays* the music"; (2) *limitation* relation, which implies that the operations are restricted by the constraints, in the same example, (constraint–operation); and (3) *characteristic* relation (constraint–object), which describes the object (that the operations can be acted upon), e.g. "*high-quality movie*".

We also identify several implicit relations (see Fig. 3.1), which include: subject–subject, subject–constraint, object–object, and constraint–constraint. These implicit relations are embedded and indirect. To elaborate these relations, we use two additional examples:

Example 18. Alice needs Bob to prove her identity so that she can play the movie.

Example 19. Alice can reuse the image in the ebook on her Web site, for educational purpose and for 2 years.

Example 18 exhibits the implicit subject–subject relation between "Alice" and "Bob", as well as implicit subject–constraint between "Alice" and "prove her identity". Example 19 exhibits implicitly the object–object relations between the "image" and "ebook", and the constraint–constraint relations between the "educational purpose" and "2 years".

3.2.3 Models

A *model* describes a typical way of using a REL; we can distinguish: (1) *revenue* model, (2) *provision* model, (3) *operational* model, (4) *contract*

model, (5) *copyright* model and (6) *security* model. A rights management system may exhibit different models simultaneously.

The *revenue* model, is normally related to the payment architecture of the system. There are myriad of revenue models, for example, pay-per-use, pay-upfront, pay-flatrate, tiered payment (e.g. free now pay later), pay to multi-entities (e.g. pay half to publisher and half to distributor), and fraction payment (e.g. discount and tax). New revenue models emerge every day.

The *provision* model may provide an *alternative solution* more than yes or no to the situations when the rights and obligations fail to meet the constraints. For instance, if viewing a high-resolution video is not allowed, it should be possible to switch to low-resolution video. Additionally, the provision model should be able to *reconcile the conflicts* caused, for example, when there is more than one subject performing the same operation on the same object simultaneously. The provision model also accommodates the *default settings of operations* over an object when the object is not associated with any operations.

The *security* model defines a variety of security mechanisms, for instance, identification, authentication and authorization (IAA), access control, non-repudiation, integrity, audit trails and privacy.

The *operational* model handles the technological aspects of the system, such as quality-of-service, watermarking, caching, network operations, bandwidth and other operational aspects of the system.

The *contract* model establishes the agreement of the terms and conditions (over the operations offered over the object and constraints) established between different subjects.

We include the *copyright* model in this category because the copyright enforcement from the user's standpoint is always a source of controversy [Camp, 2002]. The *copyright* model enforces copyright acts (especially from the end-user's standpoint), such as fair use, first sale and so on.

Not all RELs are able to support the 6 revenue models above. XrML and ODRL are not able to support the provision model of reconciling the rights conflicts. This model handles the dynamic license evolutions and content access patterns. XrML and ODRL are static RELs that are not sufficiently flexible to meet this requirement.

None of the RELs can as yet support the copyright model [Samuelson, 2003]. However, Mulligan and Burstein [2002] provide several suggestions to incorporate copyright into the XML-based RELs. We address this issue as our future work.

In the following section, we describe concisely the LicenseScript language using a simple scenario as an example.

3.3 LicenseScript Language

LicenseScript is a language that is based on (1) multiset rewriting, which captures the dynamic evolution of licenses; and (2) logic programming, which captures the static terms and conditions on a license. LicenseScript provides a judicious choice of the interfacing mechanism between the static and dynamic domains.

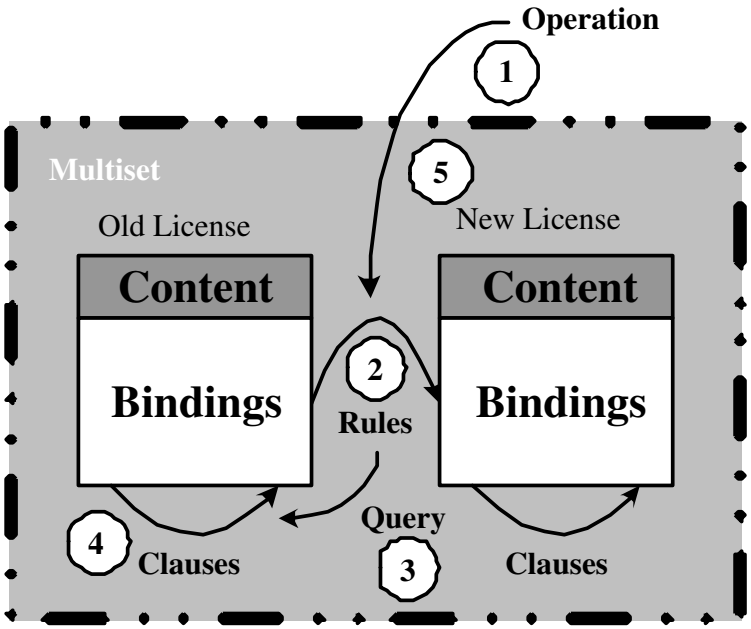

Figure 3.2: Transformation of licenses with content and bindings caused by rules.

A *license* specifies when certain operations on the object are permitted

or denied. The license is associated with the *content*, as can be seen in Fig. 3.2. A license carries *bindings*, which describe the attributes of the license; and the *clauses*, which determine if a certain operation is allowed (or forbidden). The license clauses consult the license bindings for their decision making and may also alter the values of the license bindings.

Licenses are bound to the terms that reside in multisets. For the specification of the licenses, we use logic programming. The readers are thus assumed to be familiar with the terminology and the basic results of the semantics of logic programs.

Fig. 3.2 illustrates that ① an *operation* (performed by a subject) ② invokes a *rule*, which in turns picks the required license in the multiset. The rule then generates and executes a ③ *query* on the license ④ clauses. The ⑤ execution result of the rule is a newly generated license. We elaborate this transformation process later by using a simple scenario.

Now we use a simple illustrative scenario to explain the LicenseScript:

Example 20. Amanda gets an ebook, titled "A Book" from Ben Publisher. Ben issues a license with an expiry date fixed at "23/06/2004".

This license allows Amanda to print two copies of the ebook (`L01, ..., L14` are line numbers included for reference purposes, they are not part of the code):

```
license(ebook:a_book,                    L01
[   (canprint(B1,B2,User) :-             L02
        get_value(B1,consumer,C),        L03
        C = User,                        L04
        get_value(B1,expires,Exp),       L05
        today(D), D>Exp,                 L06
        get_value(B1,printed,P),         L07
        get_value(B1,max_prints,Max),    L08
        P < Max,                         L09
        set_value(B1,printed,P+1,B2)],   L10
[   (company=ben_publisher),             L11
    (consumer=amanda),                   L12
    (expires=23/06/2004),                L13
    (max_prints=2), (printed=0)     ])   L14
```

A license is represented by a term of the form `license(content, C, B)`, where `content` is a unique identifier referring to the object of

the license; `C` is a list of *license clauses* (i.e. Prolog programs) describing under which circumstances the operations are permitted or denied; and `B` is a list of *license bindings* capturing the attributes of the license. We define two multiset-rewrite *rules*, as shown below, to model the interface between the system and the licenses. The rules can be thought of as a firmware in the user's system.

The syntax of the rules is based on the Gamma notation [Banâtre et al., 2001] of multiset rewriting (again, `R01,...,R04` are line numbers):

```
print(Ebook,User) :                          R01
    license(Ebook,C,B1) ->                   R02
    license(Ebook,C,B2)                      R03
 <= C |- canprint(B1,B2,User)                R04
```

We will step through the example assuming Amanda would like to print the eBook with the available license on her system:

1. Amanda's system wants to know whether she has the print *right* on the ebook. This is achieved by applying the print *rule* (line `R01`) with appropriate parameters: `print(ebook: a_book,amanda).`

2. The rule finds the `license(ebook:a_book,[...],[...])` (line `R02`, line `L01`) in the system. The first list refers to the license clauses (lines `L02`–`L14`), while the second list refers to the set of license bindings (lines `L15`–`L18`).

3. The rule then executes a *query* in the form of `canprint(B1,B2,User)` (line `R04`), where `B2` designates the output generated by the query; This will form a new set of license bindings.

4. The license interpreter retrieves the value of the license binding `consumer` from the list of license bindings `B1` (line `L03`) and compares the retrieved value with the user identity `User` (line `L04`). `User` is passed in as an argument to the license clause (line `L02`).

5. Similarly, the interpreter retrieves the value of the binding `expires` (line `L05`).

6. The interpreter calls the primitive (which is discussed later) `today(D)` (line `L06`) to obtain the current time and date.

7. The expiry date of the license must be greater than current time and date (line `L07`).

8. Similarly, the value of `printed` is checked if it is smaller than the value of `max_prints` (lines `L07`–`L09`).

9. If all conditions are satisfied (the user is valid, the license has not expired and the number of printed copies does not exceed the allowable maximum copies), the query returns `yes` (line `R04`) to the interpreter, with the newly generated license bindings in `B2`.

10. The value of the license binding `printed` is incremented (line `L10`) every time the print operation succeeds.

11. The value `yes` indicates that the execution of `canprint(B1,B2,User)` yields *success* in the license clauses `C`.

12. The rule `print(Ebook,User)` generates a new license with the newly generated license bindings `license(Ebook,C,B2)` (line `R03`).

The primitive `get_value(B,n,V)` is to report in `V` the value of `n` from `B`; whereas the primitive `set_value(B1,n,V,B2)` is to give value `V` to `n` in `B2`. In addition, we use a number of primitives to model the interface of the system with the license (interpreter), for instance: (1) `today(D)`, to bind `D` to the current system date/time; and (2) `identify(L)`, to identify the current environment to `L`. For further details of the LicenseScript language, see Reference [Chong et al., 2003a].

3.4 XML-based RELs Scenarios

As mentioned in section 3.1, LicenseScript is intrinsically distinct from XrML and ODRL. To help comparing them, we have studied 20 scenarios from the XML-based RELs documentation.

Table 3.1 and Table 3.2 summarizes the analysis of all the scenarios that we have studied. The (rotated) rows of the table correspond to the anatomy presented in section 3.2. The (rotated) columns correspond to the scenarios, which are:

Anatomy	**ODRL Scenarios**								**Novel Scenarios**	
Categories	E1	E2	E3	VD	SD	SW	IM	AD	PS	LM
Right-Render	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Right-Reuse			✓			✓			✓	
Right-Transport			✓		✓			✓	✓	✓
Right-Manage Object									✓	✓
Right-Regulation Right										✓
Obligation	✓	✓		✓		✓				
Object-General	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Object-Class										
Object-Delivery	✓	✓		✓		✓		✓		
Object-Fuzzy										
Subject	✓	✓	✓	✓	✓	✓	✓		✓	✓
Constraint-Temporal			✓		✓	✓	✓		✓	✓
Constraint-Bound	✓	✓	✓	✓		✓	✓			
Constraint-Environment		✓	✓			✓				
Constraint-Aspect			✓	✓						
Constraint-Purpose								✓		
Constraint-Status									✓	
Relation-Ordering-Ant. Obligat.	✓	✓		✓		✓			✓	✓
Relation-Ordering-Con. Obligat.										
Relation-Ordering-Total										
Relation-Ordering-Partial										
Relation-Association	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Relation-Naming	✓	✓	✓	✓	✓	✓	✓		✓	✓
Relation-Limitation			✓		✓	✓	✓		✓	✓
Relation-Characteristic			✓	✓						
Model-Revenue	✓	✓		✓		✓			✓	✓
Model-Provisional-Conflicts									✓	✓
Model-Provisional-Alternative										
Model-Provisional-Default										
Model-Operational										
Model-Contract	✓	✓	✓	✓	✓	✓	✓	✓		
Model-Copyright										
Model-Security-IAA	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Model-Security-Access Control									✓	
Model-Security-Confidentiality										
Model-Security-Nonrepudiation										
Model-Security-Integrity										
Model-Security-Audit Logging									✓	

Table 3.1: The properties of the usage scenarios specified in ODRL and the novel scenarios (The symbol '✓' indicates the scenario exhibits the corresponding feature).

Anatomy	**XrML Scenarios**																
Categories	PP	SB	TR	TO	PW	SL	PL	GV	SD	US	PC	WS	SW	CR	OM	SV	
Right-Render	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Right-Reuse					✓				✓								
Right-Transport							✓	✓	✓		✓						
Right-Manage Object												✓					
Right-Regulation Right											✓						
Obligation	✓	✓	✓		✓			✓	✓					✓		✓	
Object-General	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Object-Class			✓		✓												
Object-Delivery	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						
Object-Fuzzy																	
Subject	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Constraint-Temporal			✓	✓	✓	✓	✓	✓		✓	✓					✓	
Constraint-Bound											✓				✓		
Constraint-Environment				✓									✓		✓	✓	
Constraint-Aspect															✓		
Constraint-Purpose																	
Constraint-Status																	
Relation-Ordering-Ant. Obligat.	✓	✓			✓			✓	✓								
Relation-Ordering-Con. Obligat.																	
Relation-Ordering-Total			✓	✓													
Relation-Ordering-Partial				✓								✓					
Relation-Association	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Relation-Naming	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Relation-Limitation			✓	✓	✓	✓	✓	✓		✓	✓					✓	
Relation-Characteristic															✓		
Model-Revenue	✓	✓		✓	✓			✓	✓								
Model-Provisional-Conflicts																	
Model-Provisional-Alternative																	
Model-Provisional-Default																	
Model-Operational															✓		
Model-Contract	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓		
Model-Copyright																	
Model-Security-IAA	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Model-Security-Access Control																	
Model-Security-Confidentiality														✓			
Model-Security-Nonrepudiation	✓	✓		✓	✓	✓				✓	✓	✓			✓	✓	
Model-Security-Integrity				✓						✓							
Model-Security-Audit Logging					✓		✓	✓	✓	✓							

Table 3.2: The properties of the usage scenarios specified in XrML (The symbol '✓' indicates the scenario exhibits the corresponding feature).

- ODRL scenarios, which are specified in ODRL specification (`http://odrl.net/1.1/ODRL-11.pdf`):
  ebook #1 (E1), #2 (E2), #3 (E3), video (VD), super distribution (SD), software(SW), image(IM) and audio(AD);

- XrML scenarios, which are specified in XrML Use Case Examples (`http://www.xrml.org/spec/2001/11/ExampleUseCases.htm`):
  preview/promotional (PP), subscription (SB), territory restriction (TR), temporal ordering of rights (TO), usage of part of a work (PW), site license (SL), personal lending (PL) and giving (GV), super distribution (SD), unrestricted sales (US), personal copies (PC), web service access (WS), software execution (SW), confidentiality of rights (CR), operational model (OM) and secure device (SV).

- The last two columns refer to two novel scenarios, namely project documents sharing (PS) and license evolution modelling (LM), which we will discuss later.

We have translated the scenarios listed in Table 3.1 and Table 3.2 in LicenseScript. Table 3.3 shows the capabilities of XML-based RELs and LicenseScript from our studies of the usage scenarios. We put a '✓' where we can show that XML-based RELs and LicenseScript cover the corresponding feature of the REL.

Here, we list the conclusions from conducting the analysis of XML-based RELs scenarios:

1. LicenseScript can render most of the aspects of XML-based RELs, for instance, payment, user authentication etc., distinct and explicit by using the primitives and the license clauses. We use user authentication as an example: in XML-based RELs, the user identity (e.g. certificate, public key, etc.) is described by using XML tags. In LicenseScript, we use license binding to carry the user identity and use (in)equalities in the license clauses to model the user authentication explicitly.

2. LicenseScript can capture the dynamic license (along with the content) transformation caused by the operations by using the rules. For instance, we can model the transformation of an offer to a license,

Anatomy	XrML	ODRL	LS
Right-Render	✓	✓	✓
Right-Reuse	✓	✓	✓
Right-Transport	✓	✓	✓
Right-Manage Object	✓		✓
Right-Regulation Right			✓
Obligation	✓	✓	✓
Object-General	✓	✓	✓
Object-Class	✓		✓
Object-Delivery	✓	✓	✓
Object-Fuzzy			
Subject	✓	✓	✓
Constraint-Temporal	✓	✓	✓
Constraint-Bound	✓	✓	✓
Constraint-Environment	✓	✓	✓
Constraint-Aspect	✓	✓	✓
Constraint-Purpose		✓	✓
Constraint-Status			✓
Relation-Ordering-Ant. Obligat.	✓	✓	✓
Relation-Ordering-Con. Obligat.			✓
Relation-Ordering-Total	✓		✓
Relation-Ordering-Partial	✓		✓
Relation-Association	✓	✓	✓
Relation-Naming	✓	✓	✓
Relation-Limitation	✓	✓	✓
Relation-Characteristic	✓	✓	✓
Model-Revenue	✓	✓	✓
Model-Provisional-Conflicts			✓
Model-Provisional-Alternative			
Model-Provisional-Default			
Model-Operational	✓		✓
Model-Contract	✓	✓	✓
Model-Copyright			
Model-Security-IAA	✓	✓	✓
Model-Security-Access Control			✓
Model-Security-Confidentiality	✓		✓
Model-Security-Nonrepudiation	✓		✓
Model-Security-Integrity	✓		✓
Model-Security-Audit Logging	✓		✓

Table 3.3: The capabilities of XML-based RELs and LicenseScript (LS) concluded from studying the usage scenarios. The symbol '✓' indicates the REL contains the corresponding feature.

when the user *agrees* on the terms and conditions stated in offer (see section 3.5.2). We can also clearly capture the behaviour of the license evolutions when the license is copied, clipped or mixed, etc.

3. LicenseScript can model a wide variety of distinct terms in a multi-set, which could be a license, an offer, a wallet, a policy, etc. This offers a high flexibility in designing systems that are not confined to rights management. For instance, we can model a flexible payment system by using LicenseScript [Corin et al., 2003].

4. LicenseScript can provide a fine granularity of control over the content, as do XML-based RELs, as shown in Table 3.1 and Table 3.2. The tables also show that that LicenseScript is more flexible and expressive than XML-based RELs.

In subsequent sections, we use LicenseScript to describe the aforementioned novel scenarios. Thereby, we show some of the advantages of LicenseScript over XML-based RELs in addition to the listed conclusions.

3.5 Novel Scenarios

In this section, we use two novel scenarios to demonstrate the advantages of LicenseScript over the XML-based RELs.

3.5.1 Project Document Sharing

Fred, his project teammate Greg, and his project leader Han use a document sharing system for collaboration. Sometimes conflicts rise, for example, when Fred and Han are working on the same document at the same time. To reconcile these conflicts, they agree on a policy that Han's version always overwrites others' versions. Ian (from outside of the institute) has joined the project recently. Han has the privilege to grant him the system access.

The license for document "State-of-the-Art.tex", which is created by Fred, can be written as follow:

```
license(tex:state-of-the-art.tex,
[   (can_startread(B1,B2,User) :-
        get_value(B1,members,Ms), member(User,Ms),
        get_value(B1,systems,SYSs),
        identify(D), member(D,SYSs), get_value(B1,isreading,Rs),
        not(member(User,Rs)), append(User,Rs),
        set_value(B1,isreading,Rs,B2), get_value(B1,history,H),
        today(D), append([User,startread,D],H,H2),
        set_value(B1,history,H2,B2)),
    (can_endread(B1,B2,User) :-
        get_value(B1,isreading,Rs), remove(Rs,User,NR),
        set_value(B1,isreading,NR,B2),
        get_value(B1,history,H), today(D),
        append([User,endread,D],H,H2),
        set_value(B1,history,H2,B2)),
    (can_startwrite(B1,B2,User) :-
        get_value(B1,members,Ms), member(User,Ms),
        get_value(B1,systems,SYSs),
        identify(D), member(D,SYSs),
        get_value(B1,iswriting,Rs), length(RS)=0,
        append(User,Rs), set_value(B1,iswriting,Rs,B2),
        get_value(B1,history,H), today(D),
        append([User,startwrite,D],H,H2),
        set_value(B1,history,H2,B2)),
    (can_startwrite(B1,B2,User) :-
        get_value(B1,members,Ms), member(User,Ms),
        get_value(B1,systems,SYSs),
        identify(D), member(D,SYSs),
        get_value(B1,iswriting,Rs), length(RS)>1,
        get_value(B1,leader,L), User=L,
        remove(X,iswriting,NL), append(User,NL,NL2),
        set_value(B1,iswriting,NL2,B2)),
    (can_endwrite(B1,B2,User) :-
        get_value(B1,history,H), today(D),
        append([User,endwrite,D],H,H2),
        set_value(B1,history,H2,B2),
        get_value(B1,iswriting,Rs), member(User,Rs)),
    (cangrant(B1,B2,User1,User2,Sys) :-
        get_value(B1,leader,L), User1=L,
        get_value(B1,members,Ms), append(Ms,User2),
        set_value(B1,members,Ms,B2),
        get_value(B1,systems,SYSs), identify(D1),
        member(D1,SYSs), not(member(Sys,SYSs)),
        append(Sys,SYSs),
        set_value(B1,systems,SYSs,B2))  ],
[   (creator=fred), (leader=han), (members=[fred,han,greg]),
    (systems=[univ_twente]), (isreading=[]),
    (iswriting=[]), (history=[]) ])
```

The binding `members` is a list of members who have rights on this document. The binding `systems` is a list of system environments that are permitted for the members to access the document. This binding prevents the document from being accessed from an untrusted environment.

The license bindings `isreading` and `iswriting` are two sets that indicate the users who are reading and writing the document currently, respectively. In other words, they indicate the current status of the document. The license binding `history` functions as audit trail that records operations that have been performed by the users on the document.

There are 5 rules involved in this scenario, as can be seen as follows:

```
startread(Doc,User)              : license(Doc,C,B1) -> license(Doc,C,B2)
                                 <= C |- can_startread(B1,B2,User)
endread(Doc,User)                : license(Doc,C,B1) -> license(Doc,C,B2)
                                 <= C |- can_endread(B1,B2,User)
startwrite(Doc,User)             : license(Doc,C,B1) -> license(Doc,C,B2)
                                 <= C |- can_startwrite(B1,B2,User)
endwrite(Doc,User)               : license(Doc,C,B1) -> license(Doc,C,B2)
                                 <= C |- can_endwrite(B1,B2,User)
grant(Doc,User1,User2,System)    : license(Doc,C,B1) -> license(Doc,C,B2)
                                 <= C |- cangrant(B1,B2,User1,User2,System)
```

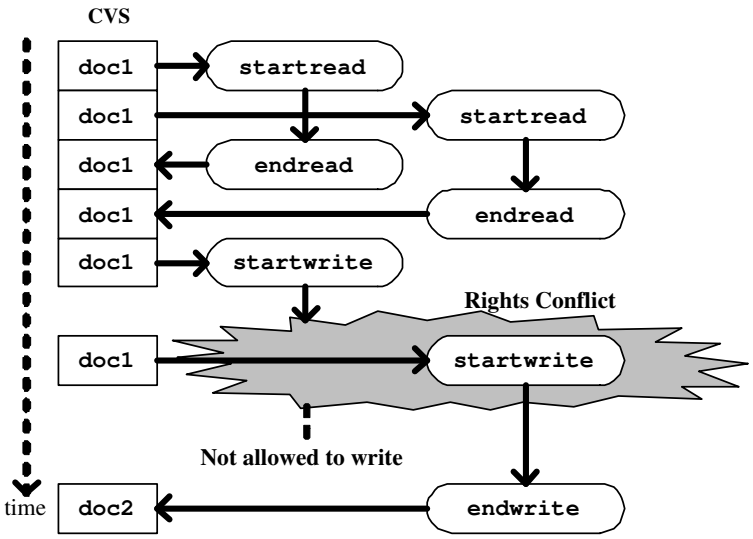

Figure 3.3: A timing diagram of rights (real-time) concurrent activities in the document sharing system.

This scenario exhibits a concurrent content access pattern: More than one subject may be performing the same operation on the same content simultaneously. This evokes conflicts.

For instance, Fig. 3.3 shows that when Fred and Han attempt to write at the same time, the project policy prevails, where it states that the project leader possesses higher privilege, Han can overwrite the document. XrML and ODRL do not provide mechanisms for concurrent access control, as far as we know. Additionally, XrML and ODRL do not provide status constraint to describe the real-time condition of the content.

3.5.2 Licenses Evolution Modelling

Jack is designing a new business which involves creating offers and licenses. He is interested in knowing how the distribution of the licenses between numerous entities would influence the licenses evolution.

He would like to check if users are allowed to assert desirable rights, e.g. a *print* right, at their systems, and he wants to know if conflicts, i.e. the asserted right conflicted with the existing right stated in the license, could arise that might undermine the business. For instance, a user asserts a new *view* right which allows her to view the ebook indefinitely.

The offer Jack constructs is:

```
offer(ebook:a_fiction,
[   (canview(B1,B2,User) :-
        get_value(B1,user,User1), User=User1,
        get_value(B1,viewed_times,PT),
        get_value(B1,max_views,Max), PT<Max, get_value(B1,expires,Exp),
        today(D), Exp>D),
    (cancopy(B1,B2,User) :-
        get_value(B1,user,User1), User=User1, get_value(B1,expires,Exp),
        today(D), Exp>D),
    (cangive(B1,B2,User1,User2) :-
        get_value(B1,user,User), User1=User, set_value(B1,user,User2,B2),
        set_value(B1,viewed_times,0,B2)),
    (canagree(B1,B2,User) :-
        today(D), Exp is D+366, set_value(B1,expires,Exp,B2),
        set_value(B1,user,User,B2)),
    (canassert(C1,C2,B1,B2,Clause,Binds,User) :-
        get_value(B1,user,User1), User=User1, append(C1,Clause,C2),
        append(B1,Binds,B2)) ],
[   (viewed_times=0), (max_views=10),
    (expires=20/12/2005), (user=anyone) ])
```

The clause `canassert` of this offer determines if the user has the privilege to add right (i.e. clause, the argument `Clause` and the bindings, the argument `Binds`) to this license. The clause `canagree` transforms

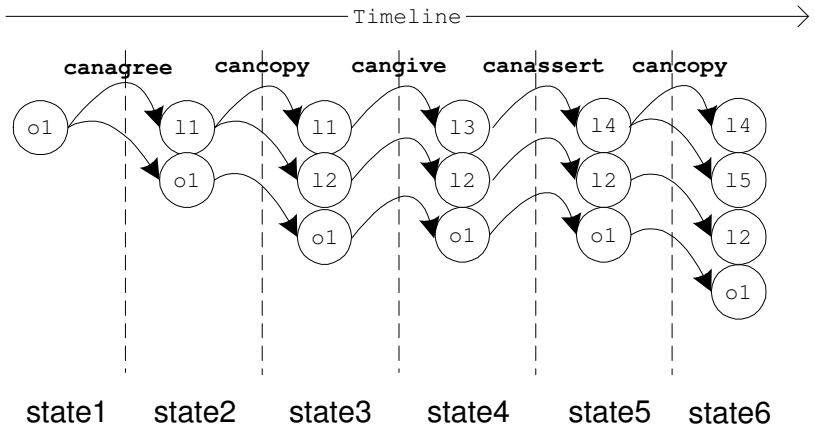

Figure 3.4: A state chart of an example of license evolution in this scenario.

this offer to a license for the user, when the user agrees on the terms and conditions stated in this offer.

Jack can use his license interpreter (simulator) to analyze the license evolutions. The license interpreter records each state of the license evolutions (i.e. each new license generated as a result of the evolution caused by the rules), including the newly generated licenses and the original licenses.

Additionally, through multiset rewriting the license interpreter is able to simulate the communications of more than two entities, for instance, content providers and content users. The license interpreter is able to help Jack tracing the logical design errors in his rights management system. Jack constructs the corresponding rules for the analysis on the evolutions of this license:

```
agree(Ebook,User)                 : offer(Ebook,C,B1) ->
                                    offer(Ebook,C,B1), license(Ebook,C,B2)
            <= C |- canagree(B1,B2,User)
view(Ebook,User)                  : license(Ebook,C,B1) ->
                                    license(Ebook,C,B2)
            <= C |- canview(B1,B2,User)
copy(Ebook,User)                  : license(Ebook,C,B1) ->
                                    license(Ebook,C,B1),license(Ebook,C,B2)
            <= C |- cancopy(B1,B2,User)
give(Ebook,User1,User2)           : license(Ebook,C,B1) ->
                                    license(Ebook,C,B2)
            <= C |- cangive(B1,B2,User1,User2)
assert(Ebook,Clause,Binds,User) : license(Ebook,C1,B1) ->
```

```
                                license(Ebook,C2,B2)
        <= C1 |- canassert(C1,C2,B1,B2,Clause,Binds,User)
```

The evolution of this offer may take the form of Fig. 3.4. The license `l4` generated (at state 5 for user `a` who asserts a *print* right on the license):

```
license(ebook:a_fiction,
[   (canview(B1,B2,User) :-
        get_value(B1,user,User1), User=User1,
        get_value(B1,viewed_times,PT),
        get_value(B1,max_views,Max), PT<Max, get_value(B1,expires,Exp),
        today(D), Exp>D),
    (cancopy(B1,B2,User) :-
        get_value(B1,user,User1), User=User1, get_value(B1,expires,Exp),
        today(D), Exp>D),
    (cangive(B1,B2,User1,User2) :-
        get_value(B1,user,User), User=User1, set_value(B1,user,User2,B2),
        set_value(B1,viewed_times,0,B2)),
    (canagree(B1,B2,User) :-
        today(D), Exp is D+366, set_value(B1,expires,Exp,B2),
        set_value(B1,user,User,B2)),
    (canassert(C1,C2,B1,B2,Clause,Binds,User) :-
        get_value(B1,user,User1), User=User1, append(C1,Clause,C2),
        append(B1,Binds,B2)),
    (canprint(B1,B2,User) :-
        get_value(B1,user,User1), User=User1)   ],
[   (viewed_times=0), (max_views=10), (expires=20/12/2006), (user=a)  ])
```

The license interpreter is able to generate all possible licenses and offers. As can be seen at state 2, the *copy* right is performed on the old license (`l1`) to generate a new license (`l2`) in addition to the original license (`l1`) at state 3.

The license interpreter can in principle be used as a licensing model checker, and the LicenseScript language *can be used as model language to specify the licensing processes*. In short, LicenseScript is a potential modelling language for licensing processes.

Additionally, LicenseScript *allows dynamic generation of new vocabulary for the rights expression in the license clause*, as shown in this scenario. XrML and ODRL do not support this feature. However, this has to be implemented with great care because this could be abusively exploited, which undermines the rights management system. We may control the rules with care, to render the rules trusted.

3.6 Conclusions and Future Work

We have presented an anatomy of right expression languages (RELs). In addition, we have studied the scenarios presented in XrML and ODRL, and we have translated them into LicenseScript (the REL we proposed in [Chong et al., 2003a]). We have also studied novel scenarios and formalized them in LicenseScript.

This investigation is useful for understanding the strengths and weaknesses of ODRL, XrML and LicenseScript, and for assessing their capability of describing a number of important content access and distribution patterns as well as licensing processes. We have also demonstrated that LicenseScript is sufficiently flexible and expressive to capture the scenarios studied so far.

In LicenseScript one can define a new vocabulary for rights expression in the license clause, which XML-based RELs cannot support (as shown in scenario of license evolution modelling, section 3.5.2). We believe that this feature of LicenseScript may support copyrights enforcement in the rights management system. This deserves further study.

CHAPTER 4

COPYRIGHT APPROXIMATION

Approximating Fair Use in LicenseScript[3]

Cheun Ngen Chong, Sandro Etalle, Pieter Hartel, and Yee Wei Law

Abstract Current rights management systems are not able to enforce copyright laws because of both legal and technological reasons. The *contract rights* granted by a copyright owner are often overridden by the users' *statutory rights* that are granted by the laws. In particular, *fair use* allows for "unauthorized but not illegal" use of content. Two technological reasons why fair use cannot be upheld: (1) the current XML-based rights expression language (REL) cannot capture user's statutory rights; and (2) the underlying architectures cannot support copyright enforcement. This paper focuses on the first problem and we propose a way of solving it by a two-pronged approach: (1) rights assertion, to allow a user to assert new rights to the license, i.e. freely express her rights under fair use; and (2) audit logging, to record the asserted rights and keep track of the copies rendered and distributed under fair use. We apply this approach in Li-

[3]This chapter has been published in 6th International Conference of Asian Digital Libraries (ICADL'2003), volume 2911 of LNCS, 2003, pages 432–443, Springer-Verlag.

censeScript (a logic-based REL) to demonstrate how LicenseScript can *approximate* fair use.

4.1 Introduction

Current rights management systems are basically not able to enforce properly copyright laws. The reason is both legal and technological and lies mainly in the fact that user's rights are a result of the reconciliation of two different and often conflicting rulings. On one hand there are the the rights granted *by contract* by the copyright owner (e.g. author or digital library) to a user; these are called *contract rights* because they are granted when user agrees on the terms and conditions imposed by the copyright owner. On the other hand, there exist *statutory rights* granted by the law.

An example of statutory right is the right of *fair use* [Mulligan, 2003]. Contract rights and statutory rights often contradict each other: a contract may for instance forbid making copies of a given book, while the law grants the user to make copies for educational use. Statutory rights depend on a number of *circumstances*. For instance, according to the United States Codes (U.S.C) (`http://uscode.house.gov/`), Section 107 Title 17 Chapter 1 (Fair Use Doctrine), "fair use of copyrighted content, including reproduction for purposes such as criticism, comment, news reporting, teaching, scholarship, or research does not violate or infringe the copyrights".

In general, statutory rights are restricted by the contract rights in the rights management systems. In other words, from the legal perspective, the copyright owner holds far more control than the copyright laws endorse [Samuelson, 2003]. Questions of the legality of overriding the statutory rights by contract rights are yet to be answered [Guibault, 2002], however the legal perspective is beyond the scope of the paper.

To fully understand why it is impossible to render this situation in current rights management systems we have to take a look at their structure; which consists of: (1) a rights expression language (REL), and (2) an underlying architecture. A REL provides a vocabulary, associated with a set of grammatical rules, to express a fine-grained usage control over a con-

tent. A *license* is written in a REL and governs the terms and conditions under which the content should be used. In practice, the most widely-used RELs are XML-based, for instance XrML [Guo, 2001] and ODRL [Iannella, 2001].

Mulligan and Burstein [2002] have pointed out the inadequacies of the aforementioned XML-based RELs in expressing a user's statutory rights: (1) the RELs can only describe contract rights; (2) the RELs provide insufficient support for rights assertion by the user; and (3) the RELs cannot provide contextual information consistent with the copyright laws that accommodate the user's statutory rights. In short, the user's statutory rights become "unauthorized" under the contract rights because they cannot be captured in the license written in an XML-based REL. On the other hand, the user's statutory rights must be upheld under the copyright laws.

Fair use allows the users to exercise these "unauthorized but not illegal" rights. In addition, it is neither a legal nor a practical requirement for users to declare these rights explicitly before enjoying these rights. Last but not least, the architecture cannot determine if some content is used for non-profit or commercial purposes [Felten, 2003]. Although it is impossible for a REL to capture the semantics of fair use completely we may *approximate* fair use [Mulligan and Burstein, 2002].

In this paper, we propose a method for implementing a digital right management system that takes into account statutory right. For this we refer to the LicenseScript right expression model [Chong et al., 2003a], and we use a two two-pronged approach based on (1) *rights assertion*; and (2) *audit logging* (see Fig. 4.1). To the best of our knowledge, this is the first attempt to approximate fair use by using a REL. We elaborate this approach in the later sections. In addition, we have pinpointed the distinction of LicenseScript with the XML-based RELs in our earlier work [Chong et al., 2003d].

This paper is organized as follows. Section 4.2 introduces our approach to approximating fair use. Section 4.3 briefly explains the LicenseScript language with a simple scenario. Section 4.4 details our approach to approximating fair use in LicenseScript. Section 4.5 describes briefly some related work. Finally, section 4.6 concludes this paper and presents future work.

4.2 Our Approach

In this section, we explain how LicenseScript may be used to approximate fair use. As mentioned in section 4.1, we are using a two-pronged approach (Fig. 4.1): (1) *Rights assertion*: to allow the user to assert new fair use-compliant rights in addition to the rights dictated by the license; and (2) *Audit logging*: to keep a record of the rights asserted and exercised by the user and to keep track of the copies of the licenses created.

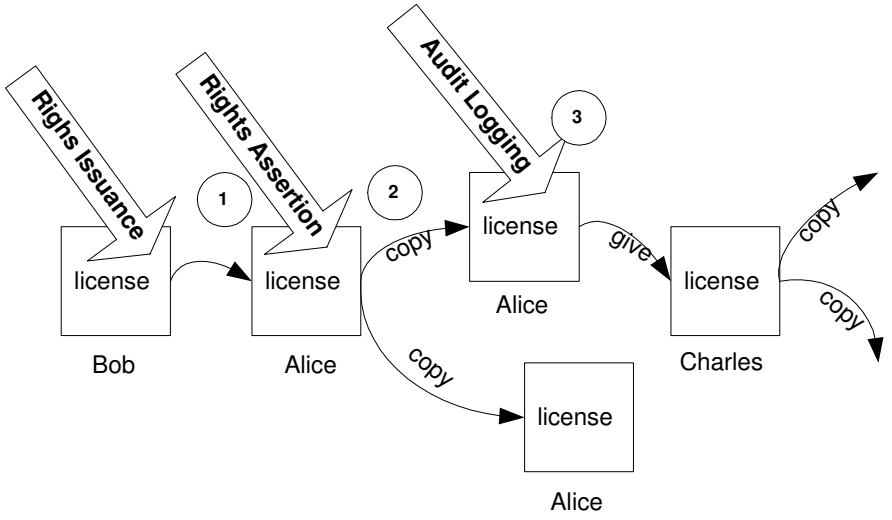

Figure 4.1: Our approach to approximating fair use.

Fig. 4.1 shows that Bob issues a license to Alice, allowing Alice to make copies of the license (and therefore she is able to make copy of the licensed content too). Alice perform rights assertion on the license before making copies of the license. Alice in turn gives a copy of the license to Charles. All the actions performed by the users (i.e. Alice and Charles) are logged in the appropriate license.

Using this approach, on one hand the users can freely exercise their statutory rights; on the other hand, the copyright owner can track the source of possible copyright infringement. Note that our proposal is more advanced than *rights issuance*, which is performed by the copyright owner for issuing and granting rights to a user.

We use the following illustrative scenario of a *digital library* to aid in our explanation in the next two sections:

Example 21. Alice borrows an ebook, entitled "An Example Book" from Bob's Digital Library (herefrom we simply call it Bob). Bob sends the license to Alice, allowing her to view, copy and give the rendered copies of the ebook to other users.

In subsequent sections, we elaborate how the two-pronged approach mentioned earlier can be used to approximate fair use.

4.2.1 Rights Assertion

Imagine a license for Alice, which states the following rights that are granted to Alice by Bob: *view*, *copy*, *give* and *assert*. Suppose Alice wants to print the ebook. The license does not state the *print* right. Therefore, Alice must assert a new *print* right by adding this right to the license. We believe that the users ability to assert new rights contributes to fair use because the user can express their rights according to their will, in addition to the rights granted by the copyright owner.

We make a few what we believe to be reasonable assumptions. Alice must have a content renderer, in this case an ebook viewer to use the ebook. A set of rules are embedded in the firmware of this ebook viewer. Bob may define these rules. Bob may not trust Alice, but he trusts the rules he defines. If Alice's asserted right in the license can be exercised by any corresponding rule Bob defines, Bob may logically trust this right (unless the asserted right causes conflicts in the license, which we will discuss later). This is because Alice's asserted rights must conform to the semantics of the rules. The implicit assumption is that the content renderer is secure.

Bob may specify some of the contextual information by using LicenseScript. These information may be, for instance, the usage purpose, the location of use etc. that the Fair Use Doctrine refers to, as discussed in section 4.1. Then, Bob can write the rules such that oblige Alice to provide the contextual information. The rules then validate the provided information using the contextual information stated in the doctrine. In other

words, Alice must declare her intention to perform fair use. The attestation of this declaration is performed by the underlying architecture (presumably by using some cryptographic means). We consider architectural support to enforce all this as our future work. Here we are concerned only with a higher level of abstraction that defines *what* may or may not happen, and not *how* actions may be performed.

4.2.2 Audit Logging

Alice should not be able to assert arbitrary rights nor must she be able to override existing rights (in the license) that may undermine the rights management system. While we might be able to avoid some problems by syntactic checks (e.g. to check for duplication of rights in the license caused by the rights assertion), many other potential ambiguities will remain (e.g. if the rights asserted can expire). Therefore, we record all the asserted rights (along with the user's identity, the date the right is asserted and the purpose of asserting the rights) in the license.

Bob may check the record and the license if the asserted rights have overridden or violated the contract rights. Therefrom, Bob may take further action, e.g. to allow/disallow the Alice's asserted right or to take Alice to court if the asserted rights violate the copyrights or the contract rights.

Additionally, Bob also tracks the copies of the licenses distributed by Alice. We can put a history record in the license to log this distribution pattern. Thus, Bob (i.e. the copyright owner) can trace the distribution of the licenses by inspecting the history record in these licenses. This helps the copyright owner track possible sources of copyrights infringement. Audit logging requires cryptographic support from the underlying architecture. We have already addressed the issue of secure audit logging in our previous work [Chong et al., 2003e].

This concludes the introduction to our two-pronged approach towards approximating fair use. We will now present the details of the approach using LicenseScript.

4.3 LicenseScript Language

LicenseScript [Chong et al., 2003a] is a language that is based on (1) multiset rewriting, which captures the dynamic evolution of licenses; and (2) logic programming, which captures the static terms and conditions on a license. LicenseScript provides a judicious choice of the interfacing mechanism between the static and dynamic domains.

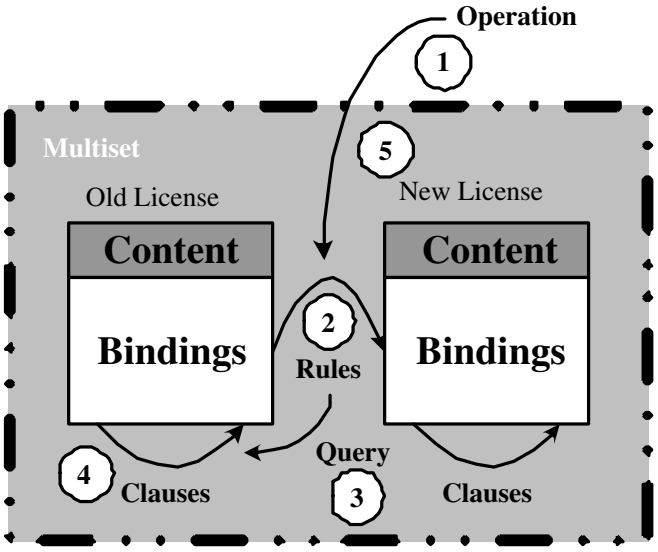

Figure 4.2: Transformation of licenses.

A license specifies when certain operations on the object are permitted or denied. The *license* is associated with the *content*, as can be seen in Fig. 4.2. A license carries *bindings*, which describe the attributes of the license; and *clauses*, which determine if a certain operation is allowed (or forbidden). The license clauses consult the license bindings for their decision making and may also alter the values of the license bindings.

Licenses are represented as terms that reside in multisets. A multiset can be thought as part of the user's system. For the specification of a license, we use logic programming. The readers are thus assumed to be familiar with the terminology and the basic results of the semantics of logic programs.

Fig. 4.2 illustrates that ① an *operation* (performed by a subject) ② invokes a *rule* in the multiset. The rule then generates and executes a ③ *query* on the ④ license clauses and bindings. The ⑤ execution result of the rule is a newly generated license. Now we use a simple illustrative scenario to explain this process:

Example 22. Amanda gets an ebook, titled "A Book" from Ben Publisher. Ben issues a license with an expiry date fixed at "23/06/2004".

This license allows Amanda to print two copies of the ebook (`L01, ..., L14` are line numbers included for reference purposes, they are not part of the code):

```
license(ebook:a_book,                        L01
[   (canprint(B1,B2,User) :-                 L02
        get_value(B1,consumer,C),            L03
        C = User,                            L04
        get_value(B1,expires,Exp),           L05
        today(D), D>Exp,                     L06
        get_value(B1,printed,P),             L07
        get_value(B1,max_prints,Max),        L08
        P < Max,                             L09
        set_value(B1,printed,P+1,B2)],       L10
[   (company=ben_publisher),                 L11
    (consumer=amanda),                       L12
    (expires=23/06/2004),                    L13
    (max_prints=2), (printed=0)     ])       L14
```

A license is represented by a term of the form `license(content, C, B)`, where `content` is a unique identifier referring to the real content; `C` is a list of *license clauses* consisting of Prolog programs describing when operations are permitted or denied; and `B` is a list of *license bindings* capturing the attributes of the license. We define two multiset-rewrite *rules*, as shown below, to model the interface between the system and the licenses. The rules can be thought of as a firmware in the user's system. The user's content renderer would contain the rules as embedded firmware. Only the copyright owner (or a trusted third party on behalf of the copyright owner) can define a set of rules for the firmware of the content renderer.

The syntax of the rules is based on the Gamma notation [Banâtre et al., 2001] of multiset rewriting (again, `R01,...,R04` are line numbers):

```
print(Ebook,User) :                          R01
    license(Ebook,C,B1) ->                   R02
    license(Ebook,C,B2)                      R03
 <= C |- canprint(B1,B2,User)                R04
```

We will step through the example assuming Amanda would like to print the eBook with the available license on her system (as shown above):

1. Amanda's system wants to know whether she has the print *right* on the ebook. This is achieved by applying the print *rule* (line `R01`) with appropriate parameters: `print(ebook:a_book,amanda)`.

2. The rule finds the `license(ebook:a_book,[...],[...])` (line `R02`, line `L01`) in the system. The first list refers to the license clauses (lines `L02`–`L10`), while the second list refers to the set of license bindings (lines `L11`–`L14`).

3. The rule then executes a *query* in the form of `canprint(B1, B2, User)` (line `R04`), where `B2` designates the output generated by the query; This will form a new set of license bindings.

4. The license interpreter retrieves the value of the license binding `consumer` from the list of license bindings `B1` (line `L03`) and compares the retrieved value with the user identity `User` (line `L04`). `User` is passed in as an argument to the license clause (line `L02`).

5. Similarly, the interpreter retrieves the value of the binding `expires` (line `L05`).

6. The interpreter calls the primitive (which is discussed later) `today (D)` (line `L06`) to obtain the current time and date.

7. The expiry date of the license must be greater than current time and date (line `L06`).

8. Similarly, the value of `printed` is checked if it is smaller than the value of `max_prints` (lines `L07`–`L09`).

9. If all conditions are satisfied (the user is valid, the license has not expired and the number of printed copies does not exceed the allowable maximum copies), the query returns `yes` (line `R04`) to the interpreter, with the newly generated license bindings in `B2`.

10. The value of the license binding `printed` is incremented (line `L10`) every time the print operation succeeds.

11. The value `yes` indicates that the execution of `canprint(B1, B2, User)` yields *success* in the license clauses `C`.

12. The rule `print(Ebook,User)` generates a new license with the newly generated license bindings `license(Ebook,C,B2)` (line `R03`).

The function `get_value(B,n,V)` is to report in `V` the value of `n` from `B`, whereas the function `set_value(B1,n,V,B2)`, to give value `V` to `n` in `B2`.

We also use a number of primitives to model the interface of the system with the license (interpreter): (1) `today(D)`, to bind `D` to the current system date/time; and (2) `identify(L)`, to identify the current environment to `L`. For further details of the LicenseScript language, see Reference [Chong et al., 2003a].

In the following section, we explain how LicenseScript can be used to approximate fair use.

4.4 Fair Use in LicenseScript

As we have seen, licenses are just objects in the multiset. Many other types of objects can be modelled, such as wallets and policies.

We use this LicenseScript-specific feature to define (1) the *license* issued by Bob to Alice, which allows her to *view*, *copy* and *give* the ebook, as well as *assert* new rights (section 4.4.1); (2) the *doctrine* that carries the contextual information consistent with fair use (section 4.4.3); (3) the *record* that logs Alice's asserted rights (section 4.4.2); and (4) the *rules* defined by Bob as the firmware of Alice's system, which include *view*, *copy*, *give*, *print* and *assert* (section 4.4.4).

4.4.1 The `license`

Following Example 21, this is the `license` that Bob issues to Alice:

```
license(ebook:an_example_book,
[ (canloan(B1,B2,Loaner,User) :-
     get_value(B1,digital_library,L), L=Loaner,
     get_value(B1,loaned,Loaned), Loaned=false,
     set_value(B1,loaned,true,B2),
     set_value(B1,user,User), today(D),
     set_value(B1,expires,D+7,B2)),
  (canreturn(B1,B2) :-
     set_value(B1,loaned,false,B2)),
  (canview(B1,B2,User) :-
     get_value(B1,user,U), U=User,
     get_value(B1,expires,Exp), today(D), D>Exp),
  (cancopy(B1,B2,B3,User) :-
     get_value(B1,user,U), U=User,
     get_value(B1,expires,Exp),
     today(D), D>Exp, get_value(B1,copies,N),
     append([(User,D),N,NN), set_value(B1,copies,NN,B2),
     set_value(B1,copies,NN,B3)),
  (cangive(B1,B2,User1,User2) :-
     get_value(B1,user,U), U=User1,
     set_value(B1,user,User2,B2), get_value(B1,trace,T),
     today(D), append([(User1,D,User2)],T,T2),
     set_value(B1,trace,T2,B2)),
  (canassert(C1,C2,B1,B2,Clause,Binds,User,Purpose) :-
     get_value(B1,user,U), U=User,
     get_value(B1,expires,Exp), today(D), D>Exp,
     get_value(B1,asserted,As),
     append([(User,D,Purpose)],Clause,NC),
     append(NC,As,As2), set_value(B1,asserted,As2,B2),
     append(C1,Clause,C2), append(B1,Binds,B2)),
  (canperform(B1,B2,User) :-
     get_value(B1,expires,Exp), today(D), D>Exp,
     get_value(B1,user,U), U=User)                    ],
[ (user=alice), (digital_library=bob), (loaned=true),
  (asserted=[]), (trace=[]),
  (copies=[]), (expires=15/8/03)                      ])
```

The function `append(L1,L2,L3)` is a built-in Prolog program that produces a new list (`L3`) by combining two lists (`L1` and `L2`).

The license clause `canloan` determines if the ebook can be loaned to the user, and *only* by the digital library. The return date (represented by `expires`) is set at the seventh day from the date this ebook is loaned. The license clause `canreturn` is the counterpart of the license clause `canloan`, which resets the binding `loaned` to `false`.

The license clause `canview` determines that *only* Alice (the user) can view the ebook and the return date `expires` has not expired. The

license clause `canassert` allows Alice to assert new rights (represented as the `Clause` with necessary bindings `Binds`). The license clause `canperform` determines if the user who performs fair use is the genuine user who owns the license.

The license binding `asserted` records all the rights asserted by the user. The license binding `trace` records the distribution of the license when it is given away. The license binding `copies` records the user who generates a new copy of this license.

4.4.2 The `record`

The `record` that belongs to Bob and which logs the rigths asserted by Alice to a license looks like this:

```
record(ebook:an_example_book,
[ (canlog(B1,B2,User,Action) :-
     get_value(B1,history,H), today(Date),
     append([(User,Action,Date)],H,NH),
     set_value(B1,history,NH,B2))              ],
[   (history=[]), (digital_library=bob)        ])
```

The term `record` records (clause `canlog`) all the actions (the argument `Action`) performed by the user (the argument `User`) at the current time (the value `Date`) on the ebook.

4.4.3 The `doctrine`

The `doctrine` (defined by Bob) that encodes the contextual information of fair use is:

```
doctrine(fairuse,
[ (canallow(B1,B2,Purpose) :-
     get_value(B1,purposes,Ps), member(Purpose,Ps),
     identify(Loc), get_value(B1,location,L), L=Loc)  ],
[ (purposes=[criticism,comment,newsreport,
             eduction,scholarship,research]),
  (location=united_states_of_america)                 ])
```

The function `member(E,L)` checks if the element `E` belongs to the list `L`. The license clause `canallow` determines if the purpose attested by the user for using the content is under the fair use context and if the user

is in the U.S. The license binding `purposes` state all the usage purposes allowed under the fair use doctrine. The license binding `location` indicates that this doctrine applies in United States. The copyright owner can define different types of `doctrine`, e.g. first sale `doctrine` to encapsulate the corresponding contextual information.

4.4.4 The rules

The rules that Bob defines for Alice's firmware are:

```
loan(Ebook,User) :
     license(Ebook,C,B1) -> license(Ebook,C,B2)
  <= C |- canloan(B1,B2,User)
return(Ebook) :
     license(Ebook,C,B1) -> license(Ebook,C,B2)
  <= C |- canreturn(B1,B2)
view(Ebook,User,Purpose) :
     license(Ebook,C,B1) -> license(Ebook,C,B2)
  <= C |- canview(B1,B2,User)
view(Ebook,User,Purpose) :
     doctrine(fairuse,Cp,Bp1),license(Ebook,C,B1) ->
     doctrine(fairuse,Cp,Bp2),license(Ebook,C,B2)
  <= C |- canperform(B1,B2,User),
     Cp |- canallow(Bp1,Bp2,Purpose)
copy(Ebook,User) :
     license(Ebook,C,B1) ->
     license(Ebook,C,B2),license(Ebook,C,B3)
  <= C |- cancopy(B1,B2,B3,User)
copy(Ebook,User,Purpose) :
     doctrine(fairuse,Cp,Bp1),license(Ebook,C,B1) ->
     doctrine(fairuse,Cp,Bp2),license(Ebook,C,B1),
     license(Ebook,C,B2)
  <= C |- canperform(B1,B2,User),
     Cp |- canallow(Bp1,Bp2,Purpose)
give(Ebook,User,Purpose) :
     license(Ebook,C,B1) -> license(Ebook,C,B2)
  <= C |- cangive(B1,B2,User1,User2)
give(Ebook,User,Purpose) :
     doctrine(fairuse,Cp,Bp1),license(Ebook,C,B1) ->
     doctrine(fairuse,Cp,Bp2),license(Ebook,C,B2)
  <= C |- canperform(B1,B2,User),
     Cp |- canallow(Bp1,Bp2,Purpose)
print(Ebook,User,Purpose) :
     license(Ebook,C,B1) -> license(Ebook,C,B2)
  <= C |- canprint(B1,B2,User)
print(Ebook,User,Purpose) :
     doctrine(fairuse,Cp,Bp1),license(Ebook,C,B1) ->
     doctrine(fairuse,Cp,Bp2),license(Ebook,C,B2)
  <= C |- canperform(B1,B2,User),
     Cp |- canallow(Bp1,Bp2,Purpose)
```

```
assert(Ebook,Clause,Binds,User,Purpose) :
     license(Ebook,C1,B1),record(Ebook,Cr,Br1) ->
     license(Ebook,C2,B2),record(Ebook,Cr,Br2)
  <= C1 |- canassert(C1,C2,B1,B2,Clause,Binds,User,Purpose)
     Cr |- canlog(Br1,Br2,User,Clause)
```

The `assert` rule says: to assert a new `Clause` with corresponding `Binds`, the user must show her identity `User` and states the `Purpose` of asserting this right; and the user's asserted clause is recorded by the term `record` (see section 4.4.2).

The first `view` rule says: to view the `Ebook`, the user must present her identity `User` and declare to the usage `Purpose`; the license must contain the license clause `canview`. If the first rule does not apply to Alice's execution, the second rule may be executed: the usage `Purpose` attested by the `User` must conform to the contextual information stated in the `doctrine` (see section 4.4.3).

The rules show how the various objects in the multiset are used, in a cooperative fashion to achieve fair use. For example, as shown in Figure 4.1, ① Alice has asserted a print right to the license (as shown in section 4.4.1) on "`1/8/2003`" for the purpose of education (by executing the `assert` rule). The new license is as follows:

```
license(ebook:an_example_book,
[ ...,
  (canprint(B1,B2,User) :-
     get_value(B1,onlyalice,U), U=User)                  ],
[ ...,
  (asserted=[
     (alice,1/8/2003,education),
     (canprint(B1,B2,User):-
        get_value(B1,onlyalice,U),U=User)]),
  (onlyalice=alice)                                      ])
```

(Herefrom, the symbol "…" represents the unchanged part of the object.)

The asserted clause `canprint` allows *only* Alice to print the ebook (she adds a new binding, namely `onlyalice` to the license). Alice can execute the `print` rule to print the ebook with the asserted *print* right. The asserted right is logged at the license binding `asserted` and Bob's `record` (from section 4.4.2) becomes:

```
record(ebook:an_example_book,
[ ...,                                                   ],
[ ...,
```

```
  (history=[(alice,
     (canprint(B1,B2,User):-
        get_value(B1,user,U),U=User),
     1/8/03)])                                          ])
```

Now, ② Alice makes an additional copy of the license by executing the `copy` rule. The third version of the license becomes:

```
license(ebook:an_example_book,
[ ...,
  (canprint(B1,B2,User) :-
     get_value(B1,onlyalice,U), U=User)                 ],
[ ...,
  (asserted=[(alice,1/8/2003,education),
     (canprint(B1,B2,User):-
        get_value(B1,onlyalice,U),U=User))]),
  (onlyalice=alice), (copies=[(alice,1/8/2003)])        ])
```

Alice's copy action is logged in the binding `copies`. ③ Alice gives a copy of the license to Charles on "`2/8/03`" (by executing the `give` rule). Charles' license (given by Alice) will look like this:

```
license(ebook:an_example_book,
[ ...,
  (canprint(B1,B2,User) :-
     get_value(B1,onlyalice,U), U=User)                 ],
[ ...,
  (asserted=[(alice,1/8/2003,education),
     (canprint(B1,B2,User):-
        get_value(B1,onlyalice,U),U=User))]),
          (onlyalice=alice),
          (copies=[(alice,1/8/2003)]),
  (trace=[(alice,2/8/03,charles)]), (user=charles)       ])
```

The distribution is logged in the license binding `trace`. The value of the binding `user` is assigned to `charles` indicating the transfer of the ownership of this license.

We have demonstrated how rules can transform the objects in the multiset by using the example as illustrated in Figure 4.1. This concludes the detailed description of the approach to approximating fair use in LicenseScript.

4.5 Related Work

Mulligan and Burstein [2002] suggest several extensions of the XML-based RELs to approximate fair use. We summarize their suggestions

as follows: (1) to define a set of rights that might simulate some "default" rights the users have with physical copies of the content, e.g. for a music album, the default rights may be *play*, *rewind*, *seek*, *excerpt* and *copy*; (2) to provide some contextual information description in the REL to support the fair use modelling, e.g. the usage purpose etc.

Similar to Mulligan and Burstein first suggestion, our approach constrains the content user's fair use rights by the firmware rules. However, the copyright owners (or content providers) cannot make the predictions on how the users would use the content. Therefore, it is a cumbersome process for the copyright owners to define a set of default rights for all available content. Our approach, on the other hand, allows user to freely express their rights. At the same time, the copyright owner may control the user's fair use actions to the extent confined by the rules. The copyright owner may flexibly define some contextual information in LicenseScript that consistent with the fair use that the rules may comply with.

Secure Telecooperation SIT, Darmstadt and the Fraunhofer Institute for Integrated Circuits, Erlangen, Ilmenau have developed the Light Weight Digital Rights Management System (LWDRM) (`http://www.lwdrm.com/`). They introduce two distinct file formats, namely local media format (LMF) and signed media format (SMF). The LMF is bound to the machine where the content is generated, whereas the SMF is intended for small-scale distribution. The SMF is generated when the user mark the content with her personal digital signature.

There are three levels of functionality defined in the LWDRM. The first level is the LWDRM player, to play the SMF/LMF content. The second level allows the user to generate LMF from the content. This level offers more extensive features to the user, e.g. improved compression algorithms etc. The third level allows the user to sign the LMF content, i.e. to generate the SMF content. Thereby, the user could distribute and use this SMF content in other machines. LWDRM may track the leak the copyright infringement by using the signature. However, the user must willingly sacrifice her privacy to perform fair use. Our approach is complementary to the SIT approach, in that we manipulate licenses while SIT manipulates content.

4.6 Conclusion and Future Work

Current rights management system can only enforce contract rights that are granted by the copyright owners to the users. Other rights, such as the statutory rights granted by copyright laws cannot be enforced in the rights management systems. Fair use is an example of statutory rights, because fair use allows "unauthorized but not illegal" actions.

In this paper, we focus on one aspect of the technological issues related to the rights expression language (REL). We argue that current RELs (1) cannot capture user's statutory rights, (2) do not support rights assertion performed by the users, and (3) cannot provide useful contextual information that is consistent with the fair use. We have introduced a two-pronged approach for approximating fair use in LicenseScript: *rights assertion* and *audit logging*. Then, we have demonstrated the use of LicenseScript to *approximate* fair use.

We would also like to investigate if LicenseScript is capable of expressing other copyright laws, e.g. first-sale doctrine as well as privacy protection in the future. In addition, we are implementing our architecture for LicenseScript, namely the LicenseScript Engine.

Part III
RIGHTS ENFORCEMENT MECHANISMS

So far, we have explained the rights expression language (REL) in general and introduced our experimental REL, LicenseScript. We have shown that LicenseScript is flexible, expressive and practical. In this part of the thesis, we will explore rights enforcement mechanisms that support LicenseScript, i.e., the mechanisms explored are capable of enforcing the rights specified by the LicenseScript license.

To enforce rights on digital content, we need to protect the content. As Lampson [1974] defines,

> *Protection*, is a general term for all the mechanisms that *control the access* of a program to other things in the system.

Many content protection mechanisms have been proposed. Some of them have been adopted as standard and deployed in our daily lives. A common example is the digital versatile disc (DVD) Content Scrambling System (CSS) [Eskicioglu and Delp, 2001]. CSS is an encryption and authentication scheme intended to prevent a DVD from being digitally copied. Additionally, the DVD can be playable according to the geographical region, i.e. to enforce region control.

Another example is PayTV, which employs a conditional access (CA) mechanism [Jain et al., 2002; Goldschlag and Kravitz, 1999]. In a CA system, the set top box in the user's home is capable of decrypting broadcast content for as long as the user remains a (paying) subscriber.

The main objective of content protection mechanisms is to prevent the digital content from being copied and distributed illegally. The basic mechanism is simple. It encrypts the content and make sure that only authorized users have the key. To implement this idea correctly is difficult, if not impossible. This is because some of the necessary equipment is entirely under control of the user, who is free to tamper with the equipment.

It is impossible to build an absolute tamper-proof system for content protection, since this must be able to withstand arbitrary, and unknown future attacks.

In response to the extreme difficulties of building an absolute tamper-proof system, the practical approach of implementing a *tamper-resistant* system has emerged as a practical substitute. A tamper-resistant system makes it *difficult* for an attacker to tamper with the protected content.

Several tamper-resistance mechanisms have been proposed. To make the thesis self-contained, we will discuss some available mechanisms in the following section.

Tamper-Resistance Mechanisms

Tamper-resistance mechanisms can be categorized into two types: *software tamper-resistance* (STR) and *hardware tamper-resistance* (HTR). We provide a brief survey here.

STR attempts to prevent or detect the content renderer from being compromised. Thereby, the attacker cannot steal or tamper with the license and content when they are being used. In addition, STR also uses techniques such as watermarking and fingerprinting to recognize specific content, when it appears in an unauthorized context. STR also tries to make it difficult to reverse engineer protected software so that embedded keys cannot be extracted easily.

Several methods to achieve a measure of STR have been proposed, such as code obfuscation [Collberg et al., 1998], code encryption [Aucsmith, 1996] and self-checking code [Horne et al., 2001a].

In addition to STR, there is work on HTR. For instance, execute-only memory (XOM) [Lie et al., 2000], trusted-computing platform(TCP) [Pearson et al., 2003], and typical hardware tokens (e.g. smart card, Java card, iButton, etc.). They strive to protect sensitive information, e.g. user's private key or software code by storing sensitive data in inaccessible parts of the hardware.

Tamper-resistance mechanisms strive to achieve the objectives discussed in the next section.

Tamper-Resistance Objectives

Ravi et al. [2004] have decomposed the objectives of tamper-resistance into more specific objectives, as shown in Figure 4.3:

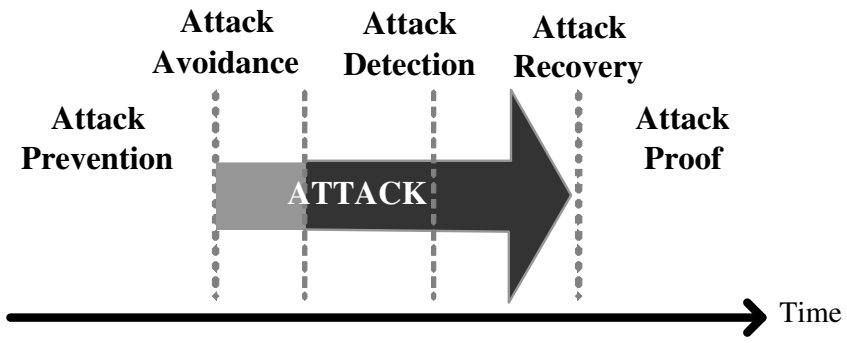

Figure 4.3: Objectives of tamper-resistant design approaches.

1. *Attack prevention* makes it difficult to launch attacks on a system. For instance, physical protection of the hardware using a stainless steel casing.

2. *Attack avoidance* [Kruegel, 2004] makes it difficult for an attacker to launch any further attacks even if she is able to access a system resource. The resource is modified in a way that makes it unusable for the attacker. For instance, the communication message between a content provider and a user is encrypted and signed.

3. *Attack detection* detects the attack after the attack is launched. For instance, an intrusion detection system. The elapsed time between the launch of the attack and the detection is critical to decide the severity of the attack. Therefore, the time delay needs to be kept as short as possible. For instance, if watermarked content is distributed illegally, the sooner the leak of illegal distribution is traced, the better the loss can be prevented.

4. *Attack recovery* returns the system to a secure state to counter the attack once it has been detected. For example, locking up or re-

booting the system so that the attack cannot further compromise the system. This will cause inconvenience to the users.

5. *Attack proof* preserves persistent records of the attack for later inspection and improvement on the system. The records can also be used for forensic purposes. Audit logging is an example of providing the attack proof (see also chapter 7).

These tamper-resistance objectives are actually closely related to each other. For instance, the unauthorized removal of an audit log (attack proof) must be detected by the system (attack detection); and the digital watermark embedded in an audio (attack detection) should not be removable (attack prevention).

In our work, we address some of the aforementioned attack models by a series of experiments: In chapter 5 [Chong et al., 2002], we model attack prevention and detection by using SPIN model, in which the thieves cannot super-distribute illegally the digital content. In chapter 7 [Chong et al., 2003e], we use secure audit logging to detect if the users have cheated in using the digital content, i.e., to achieve attack detection and proof. In chapter 8 [Chong et al., 2004], we propose a protocol working with a tamper-resistant token to protect license and metadata, which accomplishes attack prevention and detection. In chapter 9 [Cheng et al., 2004], we use tamper-resistant tokens to make it difficult for attackers stealing the digital content, i.e., to achieve attack avoidance. However, We have not deployed STR; we rely on HTR tokens. The next section summarizes this approach.

Rights Enforcement Architecture

Figure 4.4 presents the architecture for our LicenseScript implementation.

This architecture is actually refined from the digital rights management architecture proposed by Rosenblatt et al. [2002]. We separate the DRM controller into two components, i.e., License Interpreter and Reference Monitor:

- *License Interpreter* [Chong et al., 2005] interprets and maintains LicenseScript licenses. The License Interpreter supports the reference

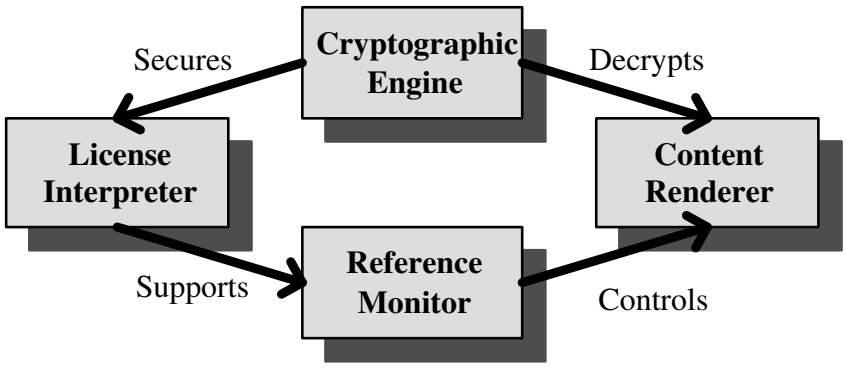

Figure 4.4: The architecture for LicenseScript.

monitor in making decisions. We describe implementation of the LicenseScript interpreter in chapter 6.

- *Cryptographic Engine* performs necessary cryptographic operations for the License Interpreter and Content Renderer. For instance, to identify, authenticate and authorize a user, to decrypt the license and content if they are encrypted, and to encrypt audit logs that record the user's actions on the content. We implement identification, authentication and authorization (IAA) on chapter 5 and cryptographic support on a hardware token in chapter 7, chapter 8 and chapter 9.

- *Content Renderer* is a customized general-purpose or special purpose application capable of rendering the content. We have customized Adobe Acrobat by building a plug-in in chapter 5. We have also created a customized audio player to play an encrypted audio in chapter 9.

- *Reference Monitor* coordinates the actions of the License Interpreter and the Content Renderer. For instance, if a user does not have a right to print a document, the print functionality of the content renderer is disabled. We further explain the concept of Reference Monitor in chapter 8.

All components of the architecture must contribute to the security of the system. For example, if the License Interpreter decides that no ac-

cess should be granted, while the connection to the reference monitor is severed, then the Content Renderer cannot be stopped. Judicious use of tamper-resistant hardware is the only option to build an appropriately secure system.

CHAPTER 5

IDENTITY–ATTRIBUTE–RIGHTS

Security Attribute Based Digital Rights Management[4]

Cheun Ngen Chong, René van Buuren, Pieter H. Hartel, and Geert Kleinhuis

Abstract Most real-life systems delegate responsibilities to different authorities. We apply this idea of delegation to a digital rights management system, to achieve high flexibility without jeopardizing the security. In our model, a hierarchy of authorities issues certificates that are linked by cryptographic means. This linkage establishes a chain of control, *identity-attribute-rights*, and allows flexible rights control over content. Typical security objectives, such as identification, authentication, authorization and access control can be realized. Content keys are personalized to detect illegal super distribution. We describe a working prototype, which we develop using standard techniques. We present experimental results to evaluate the scalability of the system. A formal analysis demonstrates that

[4]This chapter has been published in the Joint Int. Workshop on Interactive Distributed Multimedia Systems/Protocols for Multimedia Systems (IDMS/PROMS), volume 2515 of LNCS, 2002, pages 339–352. Springer-Verlag.

our design is able to detect a form of illegal super distribution.

5.1 Introduction

Annual losses to the film and music industry due to illegal distribution of content on the Internet amount to billions of dollars annually [Hartung and Ramme, 2000]. Digital Rights Management (DRM) provides a potential solution to the problem of illegal content distribution on the Internet. DRM systems manage copyrights on digital content in untrusted cyberspace.

Commercial DRM platforms for selling digital contents on the Internet are available from SealedMedia, InterTrust and Microsoft etc. Some music and movie industries have adopted online business models for instance subscription-based music sales and pay-per-view movies, such as Sony Music Entertainment and Universal Music Group. Some companies have identified the opportunity to protect intellectual property within enterprise or organization instead of business-to-consumer model, such as Authentica and Alchemedia. These systems are proprietary or include key components that are proprietary. We propose an open system, and analyse how effective it is to provide management flexibility.

Our system has components for identification, authentication and authorization to achieve a standard level of access control. However, this provides only limited flexibility, for example the in terms of the variety of permissions offered. More flexibility is needed because we cannot foresee which types of permissions and rights will be needed in future. For example, in a standard access control system when a new permission type is added to the system, the associations of all subjects and objects with their permissions must be revisited. This creates management problems. Therefore, we introduce a second level of management and control, which has a dual purpose: (1) the second level *refines* the first level control by specializing it, and (2) the second level provides facilities for the secure distribution of content. We could sloganize our contribution by: $SABDRM = AccessControl + DRM$.

Section 5.2 introduces our design of a DRM system and elaborates on the aforementioned flexibility of management with a real-life example.

Section 5.3 discusses the state of the art and related work. Section 5.4 describes our prototype system, a performance evaluation of the implementation, and a SPIN model of the system. Finally section 5.5 concludes the paper and briefly explains future work.

5.2 The Idea

We consider a client-server setting in which a user downloads content, where her rights on the content are carefully controlled, say by the content providers or by DRM service providers. We assume that each user has a unique identity. A standard public key certificate [Feghhi and Williams, 1998] can then be used to bind the identity to a public key. The public key certificates are issued and distributed by a Certificate Authority (CA) [Henry, 1999]. We assume that the server can establish the validity of the public key certificate, so that the server can identify and authenticate the user. Additionally by using the server public key certificate, the user is able to authenticate the server.

For maximum flexibility, we assume that different security attributes can be associated with each identity. Security attributes are information, other than cryptographic keys, that is needed to establish and describe the security properties of a user in the system. These security attributes may include role, group membership, time of day, location to access resources, password etc. A security attribute is encapsulated in an attribute certificate [Farrell and Housley, 2001; Gelbord et al., 2002]. The latter has a similar structure to the public key certificate but the attribute certificate does not carry a public key. The Attribute Authority (AA) [Linn, 1999], is responsible for issuing, assigning, signing, distributing and revoking attribute certificates. Note that the CA and AA could be the same authority, but for maximum flexibility, the CA and AA would be different parties. An attribute certificate is signed with the AA's private key to ensure the integrity of the attribute certificate. Having established the identity of a user, and her security attributes we are now able to decide: (a) What content is accessible to the user, and (b) What rights the user may exercise on that content.

The separation of deciding *what content is accessible* from *what can*

be done with the accessible content is a key aspect in our system; it creates flexibility, and at the same time simplifies the implementation. For example, consider a document that contains both aggregate and detailed business data. Users with appropriate security attributes may obtain the detailed data, whereas others may only obtain the aggregate data. Establishing the particular rights (i.e. the ability to view, print, save, edit etc) for any of these users holding their respective attributes is still a separate, orthogonal issue.

We use a digital license to capture the rights of a user on a particular item of content. The digital license carries information about the content, the license holder, the payment status (if payment is involved), as well as other terms and conditions of using the content. The digital licenses are described by using XrML [Guo, 2001]. XrML provides a detailed syntax for encapsulating fine-grained control information on a digital content. The Clearing House (CH) is responsible for issuing, managing payment, and distributing and revoking digital licenses. Again, the CH could be the same as one of the CA or AA but would be different from either for maximum flexibility.

The licensing mechanism we have sketched above is not vastly different from other DRM mechanism. However, there are two innovations:

1. Identity, attribute and rights are decoupled to allow for maximum flexibility.

2. Digital licenses are generated on demand at a time after the identity and the security attributes have been verified. This is the earliest moment when the system is able to decide which (partial!) content must be delivered, and what the associated rights should be.

5.2.1 Certificates and License

The difference between the public key certificate and attribute certificate and why we use a digital license can be shown from an illustration in real life here: A Malaysian resident wishes to enter The Netherlands. She would like to stay in the country for more than a year. Therefore, she needs (1) a passport, (2) a visa and (3) a residence permit.

- The Malaysian Immigration Department issues the passport. The Malaysian government has a policy to decide which of its citizens can be issued passports. This step effectively establishes the identity of a party in the system, and is a prerequisite for access control.

- The Netherlands Embassy grants the visa. The visa is a proof of permission for the passport holder to enter the country. An Embassy also works to policy to decide whom to grant a visa. This step is akin to the first level access control we mentioned in the introduction.

- The Alien Police Department in The Netherlands distributes the residence permit. The residence permit is granted according to Dutch government policies. This step corresponds to the second level access control we described in section 5.1. Three trusted authorities with three different policies are thus involved in this process.

The immigration system is manageable from the Malaysian government perspective because the Malaysian government does not have to know about Dutch government policy. The policies involved in issuing a residence permit are primarily concerned with what the visitor intends to do (study, work, vacation etc). The Malaysian government is not involved and does not even care.

A public key certificate can be seen as a passport - it identifies the owner, it tends to be valid for a long period, it is difficult to forge and it has a strong authentication process to establish the owner's identity. Our attribute certificate is like an entry visa. A different authority issues it, and in most cases, a passport has a longer validity than a visa. Consequently, acquiring the entry visa becomes a simpler and more manageable procedure. The entry visa will refer to the passport as a part of how that visa specifies the terms under which the passport owner is authorized to enter the country. Once the passport owner is identified and authenticated, the visa may authorize her to enter the country. Once in the country, the traveller may apply for a residence permit, which is then an analogue to our digital license.

5.2.2 Association of authorities

There must be a relationship between the authorities involved to establish a chain of control. For example, the visa is a stamp in the passport; therefore, the link between visa and passport is difficult to break. In the digital world, the public key certificate, attribute certificate and digital license are linked using cryptographic techniques.

An identity may be associated with several attributes, that an attribute may be associated with other attributes, and that an attribute may be associated with several rights. For example, Alice (*identity*) is an editor (*attribute*) and she belongs to an administrative group (*attribute*). The editor is only allowed to enter the system at a given time (*attribute*). The editor can print (*rights*) an annual report and she can view (*rights*) some confidential document.

Because of the interposition of attributes between the identities and the actual rights we call our system a security attribute based digital rights management (SABDRM) system. The association between these three properties identity, attribute and rights represents a chain of control. The distinctive features of the SABDRM are:

1. The use of public key certificate, attribute certificate, content identification and a source of randomness to generate a secret, unique, personalized content key. This should allow multiple use of the same key to be detected with high probability.

2. A hierarchy of authorities that is able to provide for flexibility in the rights management.

5.3 Related Work

Horne et al. [2001b] present a fair exchange protocol for peer-to-peer file sharing which encourages people to comply with the rules by providing incentives, such high quality of service, status or even air miles. Kwok and Lui [2001] have proposed a license management model to provide peer-to-peer sharing domain. They implement two types of services, one at the server side and one at the client side, to handle consumer registration,

payment, and license issuing processes and to deal with licensing in peer-to-peer distribution model.

Just regulating the distribution of digital content cannot solve the illegal distribution problems alone; it is also necessary to check that rights violations do not happen. Techniques to achieve this include watermarking and fingerprinting [Sellars, 1999]. Dittman et al. [2000] present a technology for combining collusion-secure fingerprinting schemes based on finite geometries and a watermarking mechanism with special marking points for images. Fridrich [1998] introduces the concept of key-dependent basis functions and discusses the applications to secure robust watermarking. Brin et al. [1995] describe a copy detection server which identifies copies of digital content, even for partial copies. Shivakumar and Garcia-Molina [1996] have introduced a centralized server that can detect copy or reuse (either part or whole) of digital documents. In our system, we are able to detect super distribution thanks to key personalization.

Silbert et al. [1995] propose a self-protecting container, which they call a DigiBox for protecting the digital content by providing a cryptographically protected environment for packaging content and enforcing rights. The approach proposed by Durfee and Franklin [2000] investigates trustworthiness of the distribution chain (i.e. from the Producer to the Consumers), which includes middlemen. Their approach allows trusted middlemen to alter rights but prohibits attackers from tampering with rights.

Voloshynovskiy et al. [2001] investigate various existing attacks on digital watermarking systems. They show the fundamental limits of the current watermarking technologies and argue that the present technologies are still in their adolescent stage.

Our idea of the personalized key contains the characteristic of watermarking, which embeds the user's identity, her security attribute information, the data of the content and the license. The personalized key is unique (depends on the collision-resistant one-way hash function we use to generate the key) and closely related to the content and the user. The user needs the key to access the protected content. Therefore, we believe that by keeping track of the key, we can keep track of the digital content to some extent.

Finally, yet importantly, proprietary and trustworthy client-side DRM is another problem that has been researched. The main goal is to achieve

application- and platform-independence, i.e. using different software applications to access the same protected digital content. Mourad et al. [2001] implement an application, namely WebGuard that enables existing Internet Explorer Browser and the browser's plug-ins to handle protected content. They use a technique dubbed 'Windows sub classing', which bypasses the Windows message passing within the operating system and application. We share their objective for application-independence but we work at the application level instead of the operating system level. Horne et al. [2001a] use software tamper resistance (by code obfuscation) to protect the client side. Lie et al. [2000] suggest architectural changes in the hardware that will execute only encrypted code. Their contributions have inspired our future work of applying tamper-resistant security token in a DRM system.

5.4 Prototype: SUMMER

We have developed a prototype of our DRM system in the context of the Secure Multimedia Retrieval (SUMMER) project. The main objective of SUMMER is to design a secure distributed multimedia database management system for efficient multimedia retrieval from distributed autonomous sources. Figure 5.1 shows the overview architecture of the SUMMER system. The present paper focuses on the right part. The left part (IAA-QM) is described only briefly below.

The Identification, Authentication and Authorization (IAA) module decides the identity of the user and then establishes the security attributes for that identity. Therefore, the IAA begins the chain of control of identity-attribute. As described in section 5.2, attribute certificates and public key certificates, which are generated and distributed by AA and CA respectively, are used as means to achieve IAA.

The Query Module (QM) enforces access control in the system. The QM acts as a filtering and monitoring environment on requests from the client and results from the server. The requests and the results are described using XML. A security policy file and the attribute of the client are fed in to the module. The result displays a list of digital content accessible according to the attribute of the client. The details are beyond

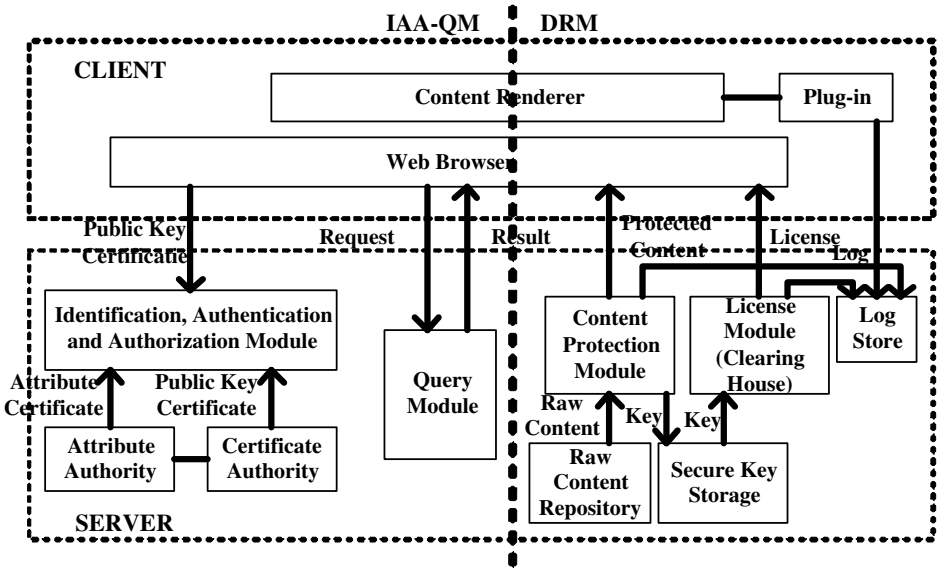

Figure 5.1: The overview architecture of SUMMER.

the scope of this paper. Further information on IAA-QM can be found in Damiani et al. [2000a,b]; Bertino et al. [1999]; Kudo and Hada [2000].

The DRM part (right half of Figure 5.1) is composed of four components, namely the Content Module, the License Module, the Content Renderer and the Plug-in. The DRM part completes the chain of control started by IAA-QM, by linking the digital rights to identity-attribute. The Content Module is composed of the following sub-components:

1. Content protection: generates, personalizes, and stores the digital content encryption key in a secure fashion. We hash the public key certificate, attribute certificate, content identity and some random data to generate a key that is unique with a high probability.

2. Raw content repository: securely stores the unencrypted digital content.

3. Secure key storage: stores the content key generated by the Content Protection module.

The License Module implements the CH (Clearing House). The License Module generates digital licenses. The module also retrieves the associated digital content key from the key storage, encrypted by using the client's public key, and embeds in the digital license. The security attributes of the user, the identity of the digital content are fed into the License Module for license generation. The License Module needs the security attributes to retrieve a list of access rights the user possesses for the content. A digital license is generated on-demand and stored. The digital license is signed by using the server's private key.

The Log Store is used to store records of all transactions. The Log Store is assumed to be secure [Schneier and Kelsey, 1998; Chong et al., 2003e]. The Log records the content protection time and license generation time on the server side as well as license interpretation process and time by the Plug-in on the client side. The Log Store provides a complete overview of all actions the client has exercised on the digital content. The purpose of the Log Store is to achieve non-repudiation, for audit trailing, and if necessary for billing. All records from the Plug-in are signed by using client's private key to prevent the client from denying any of the messages.

At the client side, a Web browser, i.e. Internet Explorer is used as an interface to communicate between the client and the server. We use Adobe Acrobat (`http://www.adobe.com`) as our Content Renderer to access the digital content in the form of PDF files. We have created a Renderer specific plug-in to interface our server with Adobe Acrobat as a proof of concept. For future work we plan to build a format/Renderer independent client side application that interfaces to arbitrary Renderers via much smaller Renderer specific plug-ins. The development of the plug-in to the Adobe Acrobat gave us first-hand experience of application customization and extension. The client side code plays an important role on:

1. Understanding and interpreting the protected digital content structure.
2. Validating the digital licenses (signature verification and time validity etc.).
3. Decrypting and retrieving the digital content key from the digital license.

4. Upholding and enforcing the client's access rights (triggering on/off the functions on the Renderer application) on the digital content according to the digital rights stated on the digital license.

5. Logs all the client's actions exercised on the digital content.

The Renderer and plug-in run in an insecure environment, which makes it possible for unencrypted content to be leaked. This problem occurs with most DRM systems. The problem can be solved to an extent by using tamper resistant hardware at the client side, and with watermarking techniques (See Section 5.3).

The security features of the prototype are as follows:

1. The digital content is kept protected (encrypted) at the client-side. Only when the user has the proper digital license and certificates, can the protected digital content be decrypted and accessed by Adobe Acrobat (using the plug-in).

2. The digital content key is personalized to the user (as described in Section 5.4). Therefore, the key can be traced back to the user. The owner of the key may thus be held responsible if a rights violation is detected – detecting such violations is outside the scope of this paper. The content key should never leave the confines of the user's machine.

3. The server (CH) is the only entity able to sign a license. Every client can verify the signature on a License.

4. Clients are not anonymous.

5. Clients can exercise rights multiple times. To control this, secure timers, counters and the like may be needed. This is an area of future work.

5.4.1 Performance Evaluation

We have performed some experiments on our content server to measure the overhead at the server side due to the on-demand content encryption

and licence generation. The measurements would allow us to determine the limitations of the server, and the extent to which the system is scalable.

We used Apache (`http://www.apache.org`) on a Pentium III 650 MHz, 128 MB RAM machine as a server. We have installed the Jakarta Tomcat (`http://jakarta.apache.org`) servlet container to interface with Apache. The client is a Pentium III 850 MHz, 256 MB RAM, which is connected to the server via an intranet 10Mbps Waveland.

The encryption algorithm we employed in the prototype is Blowfish with 128bits key [Schneier, 1994]. We implement the Blowfish encryption using the JCE library. We have varied the block size for encryption to see the possible influences on the performance. The two block sizes we have chosen are 1KB (1024 bytes) and 8KB (8192 bytes). We have not tested with larger block sizes, because a larger block size makes it harder to hide the patterns of the plaintext securely [Schneier, 1996]. Timings were generated by executing the target code block in a tight loop and by using `System.currentTimeMillis()` to capture (wall clock) timing information. We have run two sets of tests on the prototype, measuring *time as a function of content size* for content protection, as well as *time as a function of the number of users* for content protection, as shown in Figure 5.2.

A typical video is 700MB (this just fits on a 700MB CD-R); a typical MP3 is 3MB. All other digital document (TXT, PDF, DOC and so on) and digital pictures (BMP, JPEG, GIF, and so on) can be as small as 1KB, or as large as 10MB (or even bigger). To cover a significant variety of sizes of digital content, we have chosen the range 7B, 70B, 700B, ... 7MB, 70MB, 700MB. We assume a 10Mbps network (typical cable modem for home user). An MP3 can be downloaded in about 2 s, while a video takes 9 min.

The time spent downloading content is nearly a linear function of the content size. The time taken to encrypt the content with 1kB block size and 8kB block size hardly differs. We infer that varying encryption block sizes does not show a significant difference. We found that the time encryption for content size ranges from 7B to 100KB is less than 100 ms, which is imperceptible to users. The time spent for an MP3 is approximately 1 s, which is 50% of the time needed to download the content. A typical 1 hour video of 700MB takes about 5 min to encrypt; this represents a 55% time overhead. However, by overlapping encryption and

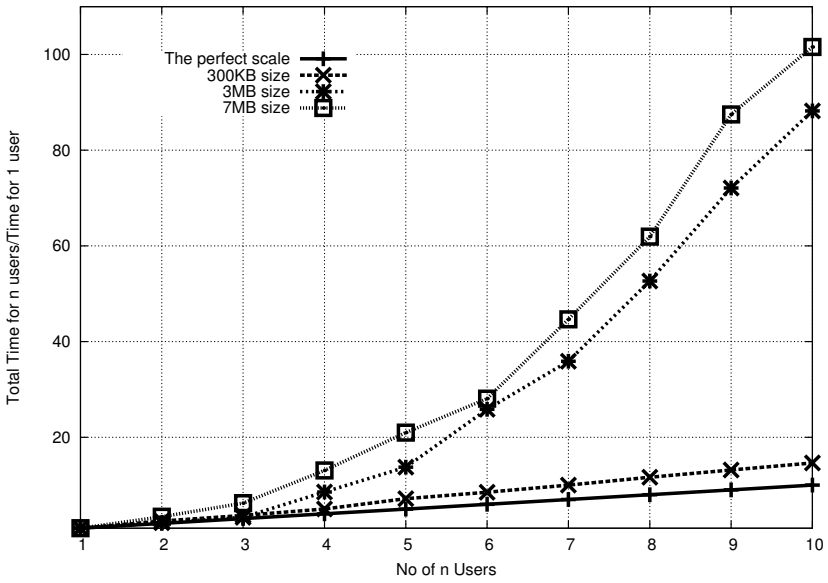

Figure 5.2: Scaled time needed for content protection as a function of the number of users activating the content encryption simultaneously.

downloading using streaming techniques [Shi and Bhargava, 1998; LaMonica, 2001], the time taken to encrypt can be completely hidden from the user.

Figure 5.2 shows how the performance of the server degrades gracefully when there is more than one concurrent user. For each of three file sizes 300KB, 3MB, and 7MB, and for each of $n = 1, 2, ...10$ concurrent users we have made a number of measurements, and plotted the average of these measurements in the figure. We have scaled the averages, dividing the average time for n users by the average time for one user. The line indicating perfect scalability is also shown. The performance of the server is not affected by small content sizes, but it becomes overloaded if the content size increases and/or if the number of concurrent users increases beyond a certain point.

To avoid degrading the service beyond the point where content encryption time can no longer hidden from the user, the server could to decide when to accept and when to reject further demands for content could on the basis of these measurements. We have evaluated the license generation

time. The size of a digital license is around 10KB, and it can be generated in 60±20ms per license. This is negligible with respect to the download times for small content size.

We conclude from the measurements that the system is scalable for digital documents, digital pictures and MP3 (of size around 3MB-7MB).

5.4.2 SPIN model

Designing correct security protocols is notoriously difficult [Lowe, 1996]. Many tools and systems are available to help the designer to analyse the protocols. Our system includes some moderately complex protocols. Therefore, we have used the SPIN model-checking tool [Holzmann, 1997] to help us explore a key property of the protocol: preventing the re-distribution of content. SPIN is able to explore the state space of a model looking for undesirable states. An example of such an undesirable state is one where content is intercepted by a Thief, and redistributed.

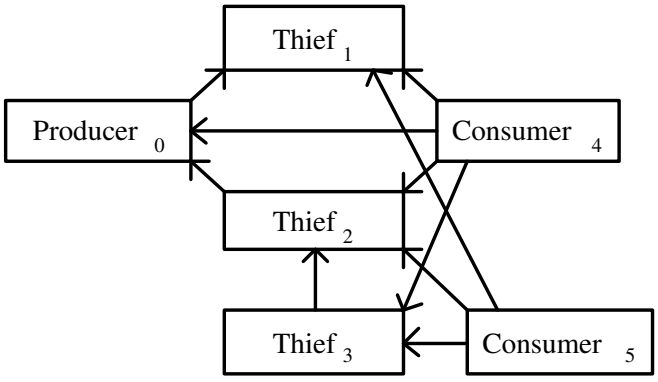

Figure 5.3: Network with a Producer, three Thieves and two Consumers showing arrows in the direction of Request message flow; License and Content messages flow in the opposite direction.

Figure 5.3 Shows a network of six processes, where the server (labelled as Producer) supports five clients. Three clients play the role of Thieves; the remaining clients (labelled Consumers) are honest, the Producer is also honest. Only the Producer is able to create original Packaged Content and Licenses. Thieves acquire Packaged Content and License

from the Producer or from each other. Thieves would do this in order to earn money: purchasing content for a certain price from the Producer, then selling it for less money to many Consumers would enable Thieves to earn arbitrary amounts of money. Consumers and Thieves can either download Packaged Content and License from the Producer or from one of the Thieves. Thief 3 demonstrates that another middleman can be involved in a transaction. Of course, there are many networks involving any number of Producers, Consumers and Thieves. We believe that the network show in Figure 5.3 represents the essence of all such networks. The processes shown in Figure 5.3 exchange three different types of messages.

- Request messages identify the desired content.
- Packaged Content messages contain the actual content encrypted under a unique content key.
- License messages contain a content key, which is signed by the Producer. The signed key is then encrypted under the public key of the process requesting the license, i.e. a Consumer or a Thief.

Request messages flow in the direction of the arrows, response messages flow in the opposite direction. In response to reach Request message, a Packaged Content and a License message are sent. For example in the simplest scenario, Consumer 4 sends a Request message to the Producer, which returns the appropriate Packaged Content and License messages to the Consumer.

To keep the model simple and yet to make it sufficiently expressive, we have made the following simplifying assumptions:

- A private key is truly private, and public keys are universally known.
- A Thief must re-encrypt a License because every client requires that a License be encrypted with her public key.
- The Producer and Consumer are honest; the Thieves are not. No parties collude.
- The Producer never reuses a content key. This allows the Consumer to check keys received for duplicates. A duplicate content key indicates the involvement of a Thief.

- Encryption is modelled by including the key in a message. Decryption then becomes comparing keys.

The model is limited and could be extended in the number of ways:

- It is possible to eavesdrop on a channel; to model this is future work.
- The Thieves are really middlemen [Durfee and Franklin, 2000] with bad intentions. A generalization to middlemen with good intentions would be valuable.
- Billing is not taken into account, this would rely on non-repudiated logging [Schneier and Kelsey, 1998].

The key property of any DRM system is to prevent illegal super distribution: Anyone other than the Producer trying to redistribute content without permission or agreement from the Producer will eventually be caught. To ensure this we rely on two assumptions: that all content keys are unique, and that customers when receiving content keys are able to check that the key received is fresh. We will show below that our SPIN model satisfies this property. The SPIN model is of course an abstraction of the real system, and as such provides no guarantees about the prototype. However, the fact that the model satisfies the prevention of illegal super distribution property demonstrates that the design of the system is sound.

We have run the model checker twice; once with the assertion by the Consumer that every key is fresh commented out and once with the assertion taking effect. No assertion violations were found with the assertion on fresh keys commented out. With be fresh keys assertion effective, a problem was found, as follows.

Firstly, a Consumer sends a Request message to a Thief, which in turn sends a Request message to the Producer. The Producer responds with Packaged Content and License message to the Thief, which caches the Packaged Content and License, then sends it along to the Consumer. The Thief takes care to re-encrypt the License with Consumers public key.

When the Consumer asks for the same content a second time, the Thief has no choice but to resend the cached Packaged Content and License again, and thus is caught. The price we have to pay for this level of security is that the Clients have to be able to cache all license keys. This requires

in principle an unbounded amount of store. However, with an appropriate hashing technique the storage requirements could be curtailed, as hashing the same key twice is guaranteed to give a collision. The question then remains how to deal with falls positives. This is an area for future research.

5.5 Conclusions and Future Work

We propose a Digital Rights Management system for multimedia content that separates the authorities involved in three categories: Identity, Attribute and Right. This separation creates flexibility because each authority has a significant degree of autonomy. The link between the authorities is established using cryptographic means to build a chain of control (*identity-attribute-rights*). Hashing the public key certificate, attribute certificate, license identity and some random data generates the content key.

We describe a prototype implementation of the system, with a performance evaluation from a users perspective. We have measured the average time needed for content encryption and license generation when simultaneously serving several users and the average time needed for content encryption while varying the size of the content. The measurements indicate that the system is scalable for digital documents, digital pictures and MP3 (of size around 3MB-7MB). A SPIN model of the system is used to validate the main protocols. Our validation result with the SPIN model asserts that super distribution is prevented when content keys are unique.

The server side DRM of our prototype is document and Renderer independent. The prototype uses Adobe Acrobat to render content. An Acrobat specific plug-in is responsible for the client side DRM. The DRM client functions as a validator for the chain of control constructed by the server, as a detector for key/license violation. The DRM client also acts as a secure store for licenses (akin to installing certificates on Web browsers).

Appendix: SPIN Model

```
/*
Verification takes about 10 mins, at 96 Mbyte, Supertrace
*/
```

```
#define NumChaches      4
#define NumServers      4
#define NumRequests     10

#define mkcontent(n)    (2*n)
#define mkkey(p, n)     (NumRequests*(p+1)+n)

#define tprocess        byte
#define tkey            byte
#define tsignature      byte
#define tnumber         byte
#define tcontent        byte


mtype = {Request, PackagedContent, License,
         NoPackagedContent, NoLicense} ;

/* Server input channels (Producer or Thieves) */
chan    ch[NumServers] = [0]
        of {mtype, tprocess, tnumber, chan, chan} ;

#define pp  0


proctype Producer(tprocess mp) {
    tnumber     n ;
    tcontent    c ;
    tkey        dk ;    /* Content encryption key */
    tkey        lk ;    /* License encryption key */
    tsignature  sk ;    /* License signature key */
    tprocess    rp ;    /* Client process id */
    chan        rod ;   /* Content output channel */
    chan        rol ;   /* License output channel */
    byte        cnt ;   /* Generate unique content key */

    cnt = 0 ;
end:do
    ::  ch[mp]?Request(rp, n, rod, rol) ;
        if
        ::  cnt < NumRequests ->
                dk = mkkey(mp, cnt) ;
                cnt++;
                c = mkcontent(n) ;
                sk = mkkey(pp, 0) ;
                lk = mkkey(rp, 0) ;

                rod!PackagedContent(n, c, dk) ;
                rol!License(n, dk, sk, lk)
        ::  else ->
                rod!NoPackagedContent(0, 0, 0) ;
                rol!NoLicense(0, 0, 0, 0)
        fi
    od
}
```

```
proctype Thief(tprocess lp; tprocess mp) {
    tnumber     n ;
    tcontent    c ;
    tkey        dk1 ;   /* Content encryption key */
    tkey        dk2 ;
    tkey        lk ;    /* License encryption key */
    tsignature  sk ;    /* License signature key */
    tprocess    rp ;    /* Client process id */
    chan        rod ;   /* Content output channel */
    chan        rol ;   /* License output channel */

    /* input channel from server */
    chan    lid = [0]
            of {mtype, tnumber, tcontent, tkey} ;
    chan    lil = [0]
            of {mtype, tnumber, tkey, tsignature, tkey} ;

    /* Cached document and license queues */
    chan    qd = [NumChaches]
            of {mtype, tnumber, tcontent, tkey} ;
    chan    ql = [NumChaches]
            of {mtype, tnumber, tkey, tsignature, tkey} ;

end:    do
    ::  ch[mp]?Request(rp, n, rod, rol) ;
        if
        ::  qd??[PackagedContent(eval(n), c, dk1)] ->
                qd??<PackagedContent(eval(n), c, dk1)> ;
                ql??<License(eval(n), dk2, sk, _)> ;
                assert(dk1 == dk2) ;
                lk = mkkey(rp, 0) ; /* (re) encrypt */
                rod!PackagedContent(n, c, dk1) ;
                rol!License(n, dk2, sk, lk) ;
        ::  else ->
                ch[lp]!Request(mp, n, lid, lil) ;
                if
                ::  lid?PackagedContent(n, c, dk1) ->
                        lil?License(n, dk2, sk, lk) ;
                        assert(dk1 == dk2) ;
                        assert(sk == mkkey(pp, 0)) ;
                        assert(lk == mkkey(mp, 0)) ;
                        qd!PackagedContent(n, c, dk1) ;
                        ql!License(n, dk2, sk, 0) ;
                        lk = mkkey(rp, 0) ; /* re-encrypt */
                        rod!PackagedContent(n, c, dk1) ;
                        rol!License(n, dk2, sk, lk)
                ::  lid?NoPackagedContent(_, _, _) ->
                        lil?NoLicense(_, _, _, _) ;
                        rod!NoPackagedContent(0, 0, 0) ;
                        rol!NoLicense(0, 0, 0, 0)
                fi
        fi
    od
```

```
}


proctype Consumer(tprocess mp) {
    tnumber     n ;
    tcontent    c ;
    tkey        dk1 ;   /* Content encryption key */
    tkey        dk2 ;
    tprocess    lp ;    /* Server process id */
    tkey        lk ;    /* License encryption key */
    tsignature  sk ;    /* License signature key */

    /* input channel from server */
    chan    lid = [0]
            of {mtype, tnumber, tcontent, tkey} ;
    chan    lil = [0]
            of {mtype, tnumber, tkey, tsignature, tkey} ;

    /* Cached document queues */
    chan    qk = [NumRequests] of {tkey} ;

end:do
    ::  /* Non-deterministic choice of document number */
        if
        ::  n = 1
        ::  n = 2
        ::  n = 3
        fi ;
        /* Non-deterministic choice of server */
        if
        ::  lp = pp
        ::  lp = 1
        ::  lp = 2
        ::  lp = 3
        fi ;
        ch[lp]!Request(mp, n, lid, lil) ;
        if
        ::  lid?PackagedContent(n, c, dk1) ->
                lil?License(n, dk2, sk, lk) ;
                assert(dk1 == dk2) ;
                assert(sk == mkkey(pp, 0)) ;
                assert(lk == mkkey(mp, 0)) ;
                assert(c == mkcontent(n)) ;
                assert(! qk??[eval(dk1)]) ; /* Comment line out */
                if
                ::  full(qk) ->
                        break
                ::  nfull(qk) ->
                        qk!dk1
                fi
        :: lid?NoPackagedContent(_, _, _) ->
                lil?NoLicense(_, _, _, _) ;
                break
        fi
    od
```

```
}


init {
    atomic {
        run Producer(pp) ;
            run Thief(pp, 1) ;
                    run Consumer(4) ;
            run Thief(pp, 2) ;
                run Thief(2, 3) ;
                    run Consumer(5) ;
    }
}
```

CHAPTER 6

LICENSESCRIPT INTERPRETER

Service Brokerage with Prolog[5]

Cheun Ngen Chong, Sandro Etalle, Pieter Hartel, Rieks Joosten, and Geert Kleinhuis

Abstract Service brokerage is a complex problem. At the design stage the semantic gap between user, device and system requirements must be bridged, and at the operational stage the conflicting objectives of many parties in the value chain must be reconciled. For example why should a user who wants to watch a film need to understand that due to limited battery power the film can only be shown in low resolution? Why should the user have to understand the business model of a content provider? To solve these problems we present (1) the concept of a packager who acts as a service broker, (2) a design derived systematically from a semi-formal specification (the CC-model), and (3) an implementation using our Prolog based LicenseScript language.

[5] A short version of this chapter has been published in the Proceedings of 7th International Conference on Enterprise Information Systems (ICEIS 2005), May 2005, pages To appear. INSTICC Press.

6.1 Introduction

A *service* is a combination of an application and its maintenance. The application implements the functionality required, e.g. making available a communication channel, playing a song. The maintenance ensures availability e.g. fast delivery, high bandwidth, 24 hour access.

Services are characterized by a wide variety of parameters [Yang and Chou, 2003], for example the capability of the service delivery (e.g. bandwidth), the flexibility of the service access (e.g. availability of the service 24 hours), and the restrictions on the service usage (e.g. device limitation). These parameters make service brokerage a complex problem.

Users have a wide variety of service requirements. For instance, they want to: have their services delivered promptly and installed properly; use their services anywhere and anytime; have a wide choice of services with various prices, qualities, etc; and they want to ensure that the technical limitations of their devices are taken into account when they acquire (purchase, lease etc) the service. In addition, users would like to protect their privacy. On the other hand, service providers have their own requirements. They need to have their services published, promoted, and more importantly paid for promptly. They also need to control the access rights of the services according to contracts established with the users. Therefore, with these different requirements from both sides of the value chain, service management becomes a complex issue.

We present the concept of a packager who acts as a service broker, and we present an implementation as part of the *Residential Gateway Environment* (RGE) project [Joosten et al., 2003; Hillen et al., 2002]. Our contribution is two-fold: (1) During the design stage, we show how to derive the complex infrastructure for the service management from a semi-formal high-level description: the "Calculating with Concept"(CC) [Dijkman et al., 2001]. We encode all aspects of service brokerage in LicenseScript [Chong et al., 2003a,b] using logic programming. (2) During the operational stage, we show efficiently LicenseScript handles the diverse requirements of all parties involved.

LicenseScript is based on Prolog and multiset rewriting and allows one to express *licenses*, i.e. conditions of use on dynamic data. Prolog has the advantage of combining an operational semantics (needed, e.g., in ne-

gotiations) with a straightforward declarative reading. Elsewhere, Prolog has also proven itself to be suitable for describing complex access policies, as demonstrated by the security language Binder [DeTreville, 2002]. Our addition of multiset rewriting to Prolog allows to encode in an elegant and semantically sound way the *state* of a license. The semantics of LicenseScript is given in terms of traces [Chong et al., 2003b]. Here, we demonstrate the practical value of LicenseScript by using it as intelligent messaging middleware for the RGE project. The result is a large distributed software platform which we describe in this paper. The platform consists of the following main components (we will elaborate these components later in section 6.4): Tomcat Server, MySQL database, JDBC database interface, etc: 50 JSP (Java Server Page) files, 20 SWF (Shockwave Flash) files. In addition we have the Prolog-based components: the ECL^{i}S^e Prolog inference engine, the LicenseScript meta-interpreter, Java user interface and RMI interfacing with RGE components: 6 Java and 2 Prolog source files.

Section 6.2 introduces the overall infrastructure of the RGE service management and its CC model. Section 6.3 derives LicenseScript from the CC model. Section 6.4 describes the RGE implementation. Section 6.5 discusses related work and the last section concludes and presents future work.

6.2 RGE Infrastructure

We present the overall infrastructure of RGE service management together with the CC method. As shown in Figure 6.1, the RGE architecture supports three main roles: the residential gateway (RG), the packager (P) and the service providers (SP).

- Service providers provide services (S), e.g. access to music, videos, but also bandwidth.
- The packager behaves as a service broker, being able to manipulate and integrate the services provided by the various SPs.
- The residential gateway is where the services actually run. A power

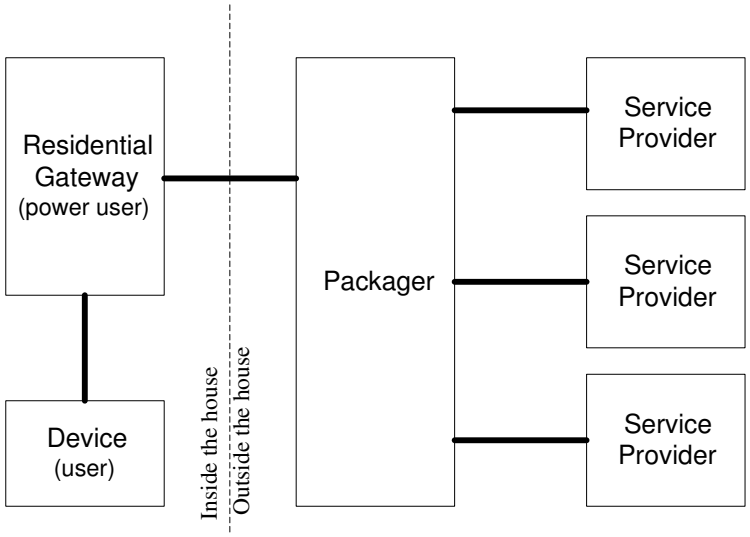

Figure 6.1: The service management architecture of the RGE.

user (PU) of the RG is allowed to (un)subscribe to services. All users (U) are allowed to use the services.

The packager has some control (on behalf of the SPs) over the services that are made available on the residential gateway. More importantly, the packager tries to match service characteristics (C) to user demands (LD). To achieve this, the packager has to have a business relation with the RG on one hand and with the service providers at the other end of the value chain.

In the rest section, we briefly introduce the CC method and we specify the RGE service management infrastructure.

6.2.1 CC Model

To develop the RGE infrastructure, den Hartog et al. [2004] use the *Calculating with Concepts (CC)* method, which can be seen as an extension of Entity-Relationship diagrams. The basic ingredients of a CC model are (a) entities, (b) relations and (c) restrictions. The rationale behind the CC-method is that every engineer involved in a project has a different in-

terpretation of the system requirements. The CC method is then used in group discussions to iron out these differences, and thus help to develop a consistent frame of reference.

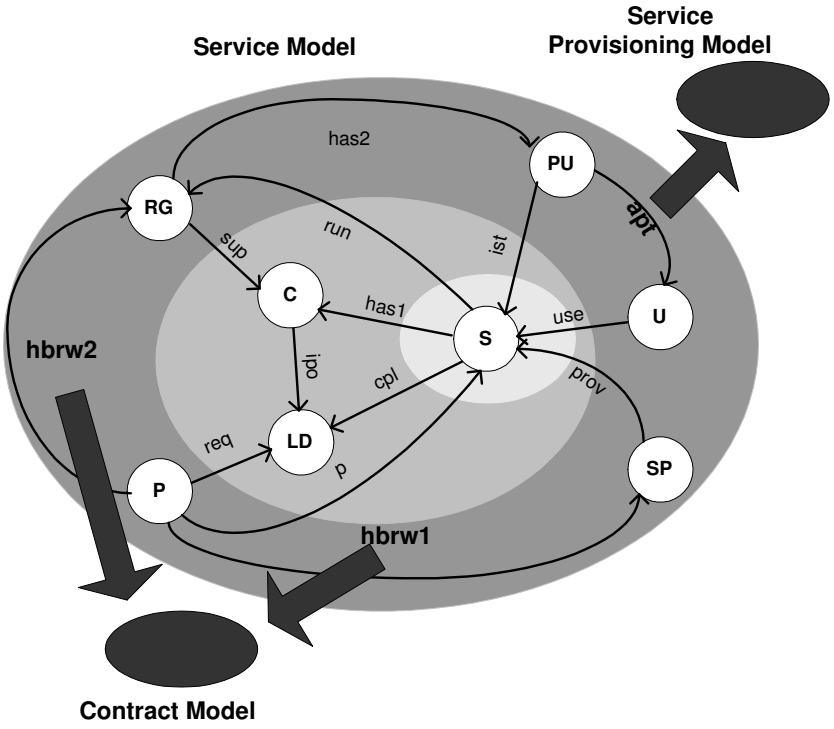

Figure 6.2: Three of the many CC models of the RGE service management infrastructure.

Figure 6.2 presents a simplified CC-model for the RGE architecture centered on service brokerage. There are many other, similar models centering on other, relevant aspects. The models are related through the use of a common vocabulary for entities, relations and restrictions.

The roles of the RGE service management infrastructure are represented by the CC entities RG, SP, PU, U, and P (see Table 6.1). These roles interact through the entities service (S), characteristics (C), and list of demands (LD). Entities, relations and restrictions are described in further detail in Tables 6.1, 6.2 and 6.3, respectively. Notice that the restrictions listed in Table 6.3 largely determine the semantics of the CC model.

For details of the CC model derivation process, the reader may refer to [Joosten et al., 2003].

Abbr.	**Entity**
C	Service characteristic, e.g. the quality, etc
LD	List of demands, which a service must comply with
P	Packager
RG	Residential gateway
S	Service
SP	Service provider
U	Normal user who uses the service
PU	Power use who possesses the administrative power on RG

Table 6.1: The CC entities of the Service Model.

Abbr.	**Relation**
apt	Power user assigns permission(s) to user.
cpl	Service complies with list of demands.
has1	Service has characteristic.
has2	Residential gateway has (is owned by) power user.
hbrw1	Packager has a business relation with service provider.
hbrw2	Packager has a business relation with residential gateway.
ipo	Characteristic is part of list of demands.
ist	Power user has subscribed to service.
p	Packager permits service.
prov	Service provider provides service.
req	Packager requires list of demands.
run	Service runs on residential gateway.
sup	Residential gateway supports characteristic.
use	User uses service.

Table 6.2: The relations between the entities of the Service Model.

#	**Cardinality Restrictions**
1	Every user is created by one and only one power user.
2	Every user has been assigned permissions by one and only one power user.
3	Every residential gateway has one and only one power user.
4	Every packager permits at least one service.
5	Every packager has a business relation with at least one service provider.
6	Every packager has a business relation with at least one residential gateway.
7	Every service is provided by at least one service provider.
8	Every service has at least one characteristic.
9	For every list of demands, there is at least one characteristic that is part of that list of demands.
10	Every packager requires at least one list of demands.
#	**Other Restrictions**
11	If a service s complies with list of demands ld, and characteristic c is part of ld, then s has c.
12	If a user u uses service s, then u has been assigned a permission by power user pu that is subscribed to s.

Table 6.3: The CC restrictions of the Service Model.

6.3 LicenseScript Derivation

We now briefly introduce the LicenseScript language, then we will show how to derive a LicenseScript specification from the CC model just presented.

LicenseScript [Chong et al., 2003a] is a formalism that can be used to specify access control and manipulation of licenses on digital content like music, video, software etc. The unique feature of LicenseScript is that licenses actually carry Prolog code (representing access and usage conditions) together with bindings, that can be used to store the *state* of the license.

In LicenseScript we work with *objects* (licenses) and *rules*. LicenseScript objects have the form:

```
object_name(Content,Clauses,Bindings)
```

Here `object_name` is the name of the object; `Content` is a content identifier which is associated to this object; `Clauses` is a set of Prolog clauses, and `Bindings` is a set of attributes pertaining to the object. Rules have the form:

```
rule_name(arguments): lhs -> rhs <== Condition
```

Here `lhs` and `rhs` are multisets of objects. `Condition` is a logical formula that may refer to the clauses defined in the objects contained in `lhs`. Because of this, rules are second-order constructs; objects are first order.

Intuitively, objects are pieces of enhanced (mobile) Prolog code, while rules are there to manipulate the objects and to query the code they carry. Rules are *not* mobile, and can be thought of as being the interface between the devices and the mobile code. Consider as an example the following rule:

```
offer(Service,S,P) :
    contracts(Service,Ccon,Bcon),
    characteristics(Service,nil,Bcha1)
->  contracts(Service,Ccon,Bcon),
    characteristics(Service,nil,Bcha1'),
    characteristics(Service,nil,Bcha2)
<=  Ccon |- canoffer(Bcon,Bcha1,Bcha1',Bcha2,S,P)
```

This rule rewrites the multiset:

```
contracts(Service,Ccon,Bcon),
characteristics(Service,nil,Bcha1)
```

into the following multiset:

```
contracts(Service,Ccon,Bcon),
characteristics(Service,nil,Bcha1'),
characteristics(Service,nil,Bcha2)
```

The proviso is that the query `canoffer(·)` succeeds when fired in the set of clauses `Ccon`.

The use of multiset rewriting allows us to model in all logical yet effective way the presence of mutable resources that not only can be modified, but also created and destroyed.

6.3.1 Deriving Licensescript

We now show how to derive LicenseScript code from the CC model. We propose a set of derivation rules to map the various CC components (i.e. entities, relations and restrictions) onto LicenseScript objects and rules, and/or the content, clauses and bindings of the objects.

1. We start from the service (entity *S*, inside the innermost circle) because this is the central entity in RGE service management. Each instance of *S* is then mapped onto the *content* part of an appropriate LicenseScript object.

2. Entities are split into two groups:

   - Those that have a *direct* relationship with *S* (*C* and *LD* in the middle circle) are mapped into LicenseScript *objects*.
   - Those that have an *indirect* relationship with *S* (*RG, P, SP* and *U*, in the outer circle) are mapped into LicenseScript *bindings*.

3. Relations between the entities are mapped onto clauses, the body of which must reflect the cardinality restrictions of the relation.

4. Other general CC restrictions are captured by LicenseScript multiset rewrite rules.

LicenseScript Object	Description
`characteristics(S,nil,B)`	Represents characteristics of a service.
`demands(S,C,B)`	Represents list of demands required.
`license(S,C,B)`	Represents permissions/rights.
`contracts(S,C,B)`	Represents business relations.

Table 6.4: The LicenseScript objects representing CC entities, where `S` represents the service; `C` denotes a set of clauses; and `B` is a set of bindings.

6.3.1.1 Objects

Objects are the result of mapping entities of the middle circle: *LD* becomes `demands(S,C,B)` while *C* is mapped onto `characteristics (S,C,B)`. In addition, to communicate with the external world, (Contract Models and Service Provisioning Models, in Figure 6.2), we use the objects `license(S,C,B)` and `contracts(S,C,B)`. Table 6.4 provides a summary. The reader may refer to the technical report [Joosten et al., 2003] for more information.

6.3.1.2 Clauses

In principle, derivation rule #3 maps each cardinality relation onto a separate clause. To improve efficiency, we map more than one CC relation onto a single clause. For instance, we use the clause `cangrant(·)` to capture both relations *has2* and *apt*. This clause allows the power user to assign the license (i.e. grants the usage permissions) to normal users:

```
cangrant(Blic1,Blic1',Blic2,Poweru,User) :-
    get_value(Blic1,power_user,Pu),
    authenticate(Pu,Poweru),
    set_value(Blic1,user,User,Blic1').
```

Here `Blic` and `Blic1'` are `bindings`. To access these bindings we use the primitives below to get (resp. set) the value associated with `Name` in `Bindings`:

```
get_value(Bindings,Name,Value)
```

```
set_value(Bindings,Name,Value,NewBindings)
```

As a second example, the clause `canuse(·)` allows to capture the relations *run* and *use*. `canuse` authenticates and checks that the service actually runs on the residential gateway (the binding `enabled`):

```
canuse(Blic,Blic',User) :-
    get_value(Blic,user,U),
    get_value(Blic,enabled,E),
    E == true, authenticate(U,User).
```

We conclude this part by showing a more complex example. `canpermit(·)` authenticates the packager to ensure its genuineness, before enabling the service to the residential gateway; relations *p* and *ist* are captured here.

```
canpermit(Bdem,Bdem',Blic,Clic,Pack) :-
    get_value(Bdem,packager,P),
    get_value(Bdem,service_provider,S),
    get_value(Bdem,license_clauses,Clic),
    authenticate(Pack,P),
    set_value(Bdem,enabled,true,Blic).
```

The parameter `Clic` allows to extract a whole new set of clause from the bindings and to create a new LicenseScript object with it.

Other clauses are derived in the same way from the CC model. A summary of the definitions is given in Table 6.5.

Clause	Object	Relations	Restrictions
`cancomply(·)`	`demands(·)`	has1, cpl, ipo, sup	8, 9, 10
`canpermit(·)`	`demands(·)`	p, ist	4
`canoffer(·)`	`contracts(·)`	prov	5, 7
`canrequest(·)`	`contracts(·)`	req	6
`cangrant(·)`	`license(·)`	has2, apt	2, 3
`canuse(·)`	`license(·)`	use, run	1

Table 6.5: The LicenseScript clauses that capture the relations in Table 6.2 and the conditions of success for the restrictions in Table 6.3.

6.3.1.3 Rules

Rules provide the necessary interface between the outside world and the LicenseScript objects. The simplest example of rule is `use`, which is invoked by the user to actually use a service. The rule just has to check for the presence of a license:

```
use(Service,U) :
    license(Service,Clic,Blic1)
->  license(Service,Clic,Blic2)
<=  Clic |- canuse(Blic1,Blic2,U)
```

`canuse(Blic1,Blic2,U)` is queried in `Clic` (a failure of the query would indicate that the license is no longer valid; e.g. it might have expired); after successful completion of the query the license is replaced by another one with a the new set of bindings `Blic2`.

A more complex rule is `grant`, which duplicates a license. The power user would execute `grant` to grant some permissions/rights to a normal user:

```
grant(Service,U1,U2) :
    license(Service,Clic,Blic1),
->  license(Service,Clic,Blic1'),
    license(Service,Clic,Blic2)
<=  Clic |- cangrant(Blic1,Blic1',Blic2,U1,U2)
```

This rule generates a new `license(·)` for the user.

Finally we present the rule `permit`, with which the packager generates a license for some service to be run on the residential gateway:

```
permit(Service,P,S) :
    demands(Service,Cdem,Bdem),
    characteristics(Service,nil,Bcha)
->  demands(Service,Cdem,Bdem')
    characteristics(Service,nil,Bcha'),
    license(Service,Clic,Blic)
<=  Cdem |- canpermit(Bdem,Bdem',Blic,Clic,P),
    Cdem |- cancomply(Bdem,Bdem',Bcha,Bcha')
```

The object `demands(Service,Cdem,Bdem)` indicates that a user has requested `Service`; `Cdem` and `Bdem` are respectively a set of clauses

and a set of bindings that – combined – specify extra side conditions such as the maximum bandwidth, the price the user is willing to pay, etc. By calling `canpermit`, the packager checks if permission can be granted. `canpermit` also returns the clauses that will be used in the new license. On the other hand, `cancomply` validates the service request (See below).

6.3.2 Service Requirements Validation

We now define how the packager validates a service request, first using a simplified version of the `cancomply(·)` clause:

```
cancomply(Bdem,Bdem',Bcha,Bcha') :-
    get_value(Bdem,bandwidth,X1),
    get_value(Bcha,bandwidth,X2),
    get_value(Bdem,quality,Y1),
    get_value(Bcha,quality,Y2),
    get_value(Bdem,billing,Z1),
    get_value(Bcha,billing,Z2),
    X1 >= X2, Z1 = Z2, Y1 =< Y2.
```

The last line shows the use of constraints to ensure that the maximum bandwidth of the user's device meets the minimum bandwidth required for the service; that the billing status of the user meets the requirement of the service provider; and that the quality measure required by the user does not exceed the offered quality.

Alternatively, one can use a parametric approach, in which the list of requirements to be complied with is stored in the license:

```
cancomply(Bdem,Bdem',Bcha,Bcha') :-
    get_value(Bcha,requirements,Requirements),
    meets_requirements(Requirements).

meets_requirements([]).
meets_requirements([[Req_name,Req_value]|Reqs]):-
    check_requirement(Req_name,Req_value),
    meets_requirements(Reqs).
```

Recall that (see rule `permit` above) the query `cancomply` is fired in the set of clauses `Cdem` specified in the user's demand `demands(`

`Service, Cdem, Bdem)`. Therefore `cancomply` can check that the service specification meets the the constraints set out in the user's demand.

Related to this, Corin et al. [2003] have demonstrated that LicenseScript allows one to define flexible payment policies that may be set by users. This is non-trivial as there may be more than one service provider interacting with one packager:

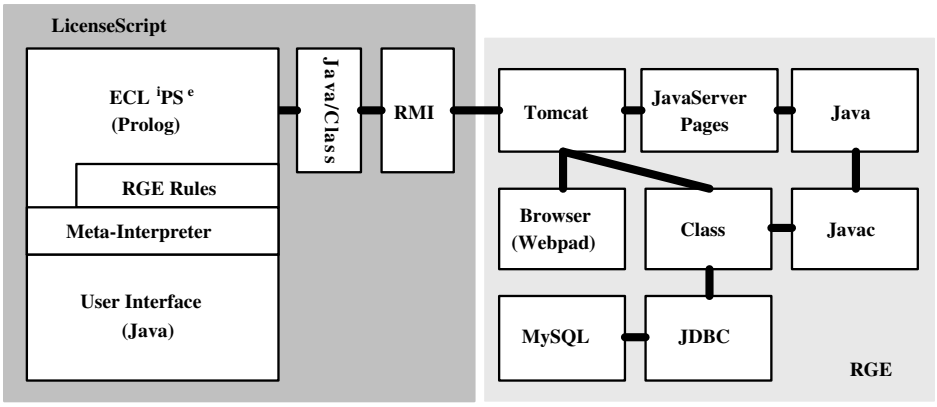

Figure 6.3: Configuration of the RGE demonstrator software components.

6.4 RGE Demonstrator

We have finally come to the RGE demonstrator proper. As shown in Figure 6.3, the demonstrator contains a number of off-the-shelf software components. On the right hand side we find among others the Tomcat server and the MySQL database.

The left hand side of Figure 6.3 zooms in on the specific LicenseScript components. The basic one is the LicenseScript meta-interpreter, which is implemented using ECL^{i}PS^e (`http://www.icparc.ic.ac.uk/eclipse/`).

The core of the LicenseScript meta-interpreter is a simple extension of the vanilla meta-interpreter presented in [Sterling and Shapiro, 1994]:

```
solve([],_).
solve([Query|Queries],Program) :-
```

```
    copy_term(Program,Temps),
    member((Query:-Body),Temps),
    solve_body(Body),
    solve(Queries,Program).

solve_body(true).
solve_body((Goal1,Goal2)) :-
    solve_body(Goal1),solve_body(Goal2).
solve_body(Goal) :-
    call(Goal).
```

The only difference lies in the fact that LicenseScript runs a query in a specific `Program`. Multiset rewriting has also been implemented in Prolog [Chong et al., 2003a].

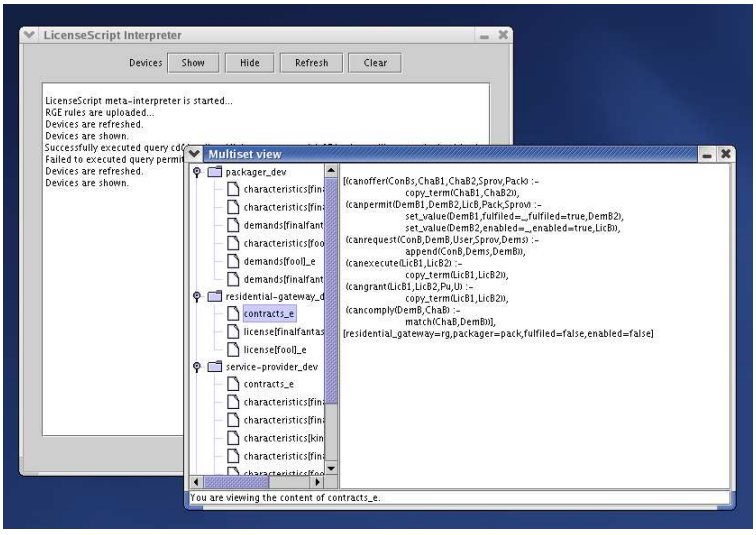

Figure 6.4: The LicenseScript Interpreter user interface.

The meta-interpreter is interfaced with the Tomcat server via a Java interface, which is called by remote method invocation (RMI).

Figure 6.4 reports a snapshot of the (Java) user interface for the LicenseScript components. Various buttons allow the user to view the LicenseScript objects in the various devices of the multiset. The text area underneath the buttons logs and displays the status of the execution of the

meta-interpreter and the ECL^{i}PS^e engine. The multiset viewer allows the users to observe the status of the LicenseScript objects before and after the execution.

As shown in Figure 6.5, when the demands and the characteristics of the service do not match, a dialog notifies the packager.

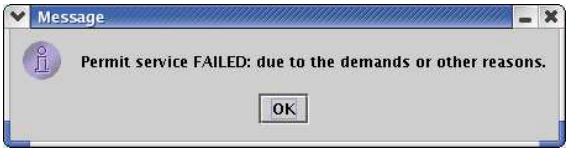

Figure 6.5: A dialog showing that the demands and the characteristics of the service do not match.

To conclude this section, we show a snapshot of the RGE demonstrator in Figure 6.6. The leftmost Web browser represents the service provider, the middle is the packager and the rightmost is the residential gateway.

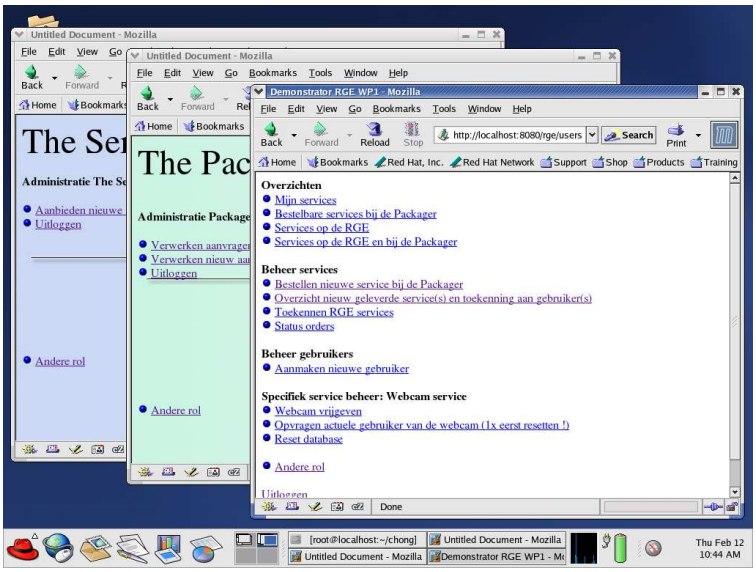

Figure 6.6: The RGE demonstrator.

6.5 Related Work

We discuss related work on Web service management and service brokerage in section 6.5.1 and section 6.5.2, respectively.

6.5.1 Web Service Management

Web services are services provided over the Internet. Web services are described using the XML-based Web Services Description Language (WSDL) (`http://www.w3.org/TR/wsdl`). WSDL provides a simple way for service providers to describe the *basic format of requests* (made by the users) to their systems regardless of the underlying protocol and the message encoding format. The underlying protocol of WDSL is the Simple Object Access Protocol (SOAP) (`http://www.w3.org/TR/SOAP/`). SOAP is used to describe the envelope and message format. In addition, SOAP provides a basic request/response handshake protocol that for exchanging structured information. WSDL/SOAP have emerged as a standard for handling Web service architecture. They have been deployed in some current Web service frameworks, e.g. IBM WebSphere (`http://www.ibm.com/websphere`) and Microsoft .Net (`http://www.microsoft.com/net`).

WSDL describes the Web services merely as a set of end-points (of the connections) operating on the messages. This makes it difficult to match parameterised service requests to offerings because matching requires flexibility in making the connections. This is precisely what LicenseScript is good at. On the other hand LicenseScript provides no facility for the description of message transmission format and message exchange protocol, as WSDL/SOAP does; these facilities are beyond the scope of LicenseScript.

As mentioned earlier, there exist several frameworks for Web service management, e.g. Java 2 Platform Enterprise Edition (J2EE) (`http://java.sun.com/j2ee`) (which is integrated in the IBM WebSphere) and Microsoft .Net. These frameworks provide facilities for *service description*, *service implementation*, *service publishing, discovery and binding*, as well as *service invocation and execution* [Fletcher et al., 2003]. In LicenseScript, we have concentrated on service description. Actually, the overall RGE project has addressed the service implementation, service in-

vocation and execution.

6.5.2 Service Brokerage

We have seen how LicenseScript can be used for *service brokerage*. We now discuss related work on this topic.

Bichler and Segev [1999] present a framework for service brokerage, where the broker is represented as an *agent*. Typically, an agent is programmed to search and aggregate the information from the Internet. An advanced agent is able, for example, to perform price comparison on some products (e.g. BargainFinder [Greenwald and Kephart, 1999]); or to negotiate and participate in an online auction on behalf of its owner [Guttman and Maes, 1998].

While agents provide an elegant and convenient way to model tasks such as service brokerage, their deployment can raise privacy concerns and security problems [Chess, 1998]: agents need to be protected against possible malicious host, and vice versa.

Our licenses are mobile and contain code (Prolog clauses) and data (Bindings); in this sense licenses can be interpreted as agents. However, we make a number of assumptions to alleviate security problems. In particular we assume that the rules and the prolog engine can be trusted (they represent the firmware of our devices).

6.6 Conclusions and Future Work

We present one of the central concepts of the LicenseScript-RGE demonstrator, i.e. the *packager*, who acts as a service broker. We derive its implementation in our Prolog based LicenseScript language, using a systematic derivation from a semi-formal specification (the CC-model).

Prolog proves to be a very suitable platform for implementing a complex broker such as the one we have presented, in particular:

- To represent complex services in a flexible and efficient manner one needs to employ executable (mobile) code of some kind. To manipulate services it is therefore necessary to employ a second-order system. Prolog is perfect for this.

- Services should not only be executable, but should have a clear and concise semantics (after all, they are *licenses*). The close relation between operational and the declarative semantics of Prolog is an invaluable advantage.

- Prolog is ideal to match requirements, and good at resolving conflicts. Therefore it is a natural platform for service brokerage.

Thanks to Prolog's expressive power – the LicenseScript engine consists of just a few dozens of lines of code. Also services (which are represented as objects, containing Prolog code), usually require only few Prolog lines to be described.

In the future, we are planning to implement the concept of authorized domain [van den Heuvel et al., 2002] in the RGE. We would also like to enhance the security of LicenseScript objects by using some tamper-resistant hardware [Chong et al., 2003e].

CHAPTER 7

SECURE AUDIT LOGGING

Secure Audit Logging with Tamper-Resistant Hardware[6]

Cheun Ngen Chong, Zhonghong Peng, and Pieter H Hartel

Abstract Secure perimeter schemes (e.g. DRM) and tracing traitor schemes (e.g. watermarking, audit logging) strive to mitigate the problems of content escaping the control of the rights holder. Secure audit logging records the user's actions on content and enables detection of some forms of tampering with the logs. We implement Schneier and Kelsey [1998] secure audit logging protocol, strengthening the protocol by using tamper-resistant hardware (an iButton) in three ways: Firstly, our implementation of the protocol works offline as well as online. Secondly, we use unforgeable timestamps to increase the possibilities of fraud detection. Lastly, we generate the authentication keys, core security of Schneier and Kelsey's protocol on the iButton to alleviate the possibilities of malicious client generating the bad keys. We provide a performance assessment of our im-

[6]This chapter has been published in 18th IFIP International Information Security Conference (IFIPSEC), volume 250 of IFIP Conference Proceedings, 2003, pages 73–84. Kluwer Academic Publishers.

plementation to show under which circumstances the protocol is practical to use.

7.1 Introduction

Digital content is so easily distributed, and dissociated from the metadata that describes owner, terms and conditions of use etc. that copyright infringement is rife. Secure perimeter schemes such as digital rights management (DRM) alleviate the problem in some cases [Chong et al., 2002] but most (if not all) DRM systems are vulnerable to attacks. The raw content can then be redistributed, severely damaging the interests of the rights holder. Tracing traitor schemes trace leaks of content to the users who can be identified, and ultimately whose behaviour can be recorded as evidence. Many techniques exist to rediscover the identity and thence the rights on the content, such as cryptography, digital fingerprinting, watermarking etc. In this paper, we assume that users can be identified, and we concern ourselves with the issue of gathering information on the user's behaviour.

Secure audit logging records the actions of a user on an item of content and does so in a manner that allows some forms of tampering with the log to be detected. We implement Schneier and Kelsey [1998] secure audit logging protocol, using tamper-resistant hardware (TRH). For brevity in the sequel, we refer to Schneier and Kelsey as "SK".

An audit log is an important tool to detect and to comprehend damages of a computer or network system caused by intrusions, defects or accidents. An audit log contains descriptions of noteworthy events. In our DRM experiment audit logs are generated in the user's personal computer (PC). The PC is a hostile environment (untrusted domain) because of its vulnerability against various malicious attacks. Therefore, the audit logs require protection to ensure integrity.

SK involves two parties: an untrusted machine and a trusted machine. The untrusted machine is not physically secure or sufficiently tamper-resistant. SK makes the audit logs survive the adversary's attacks. In other words, SK renders audit logs impossible for an adversary to *undetectably* view, forge and delete even after the untrusted machine is compromised by

the adversary. Furthermore, the audit logs record all the actions performed by the adversary, including her attempts to compromise the untrusted machine.

The comment by SK that "the trusted machine may typically be thought of as a server in a secure location, or implemented in various ways, which includes a tamper-resistant token" has inspired us to use tamper-resistant hardware (TRH) as the trusted machine for secure logging. The TRH (or the trusted machine) we use is a Java iButton (`http://www.ibutton.com`) for the following reasons:

1. The iButton contains a programmable tamper-evident real time clock. The real time clock keeps time in $\frac{1}{256}$ second increments.
2. The iButton supports efficient implementations of common cryptographic algorithms.
3. The iButton version 1.1 provides up to 6kB non-volatile RAM, the more expensive version 2.2 contains approximately 134kB non-volatile RAM.

We use the iButton as a trusted device to aid in the audit log creation in the manner proposed by SK. An iButton is too small to store a log of any useful size in a cost effective manner: A typical PC contains 40GB storage, i.e. around $300,000$ times more than the iButton.

Our DRM system has the usual Client/Server architecture. The Client is a user with her PC, which represents the untrusted domain. The Server is a trusted environment where content and license are stored. When the Client accesses the content piecemeal from the Server (e.g. by streaming), the latter is able to protect the content to some extent because the Client's actions can be monitored. However, when the Client downloads the content to the PC's non-volatile storage to accesses the content offline (i.e. disconnected from the Server), the Server is not able to monitor the Client's behaviour. We propose using secure audit logging with TRH to bring the security of offline DRM to the level of online DRM. The main contributions of this paper are:

1. To implement SK embedded in several auxiliary protocols and with the iButton, to support security of offline DRM.

2. To evaluate the performance of the implementation; thus investigating whether the iButton can be used effectively.

3. To strengthen SK by making sure that some of its security assumptions are valid by virtue of using the iButton. We generate core secrets and timestamps on the iButton instead of the untrusted PC.

To the best of our knowledge ours is the first attempt to implement SK and the first endeavour to analyse the performance of SK in general, and SK with iButton in particular.

A weakness of any system, which relies on TRH to coerce an untrusted Client into specific behaviour, is that the user may simply sever the connection between the Client and the TRH. We suggest a number of ways to discourage the Client from such behaviour: (1) The Server is designed so that it insists on the iButton being present to authenticate and authorize the Client; (2) Organizational policy (e.g. in a corporate intranet) is used to enforce the use of the iButton. In both cases users are provided an incentive to maintain communication with the iButton: *no iButton means no content*.

The main problem with audit logging is that at some stage a dishonest user may cheat by disabling the audit logging functionality of the DRM application at the Client. She is able to deny any actions she has performed during the offline period without evidences in the log.

At this point, the protection offered by DRM is weak on PCs but potentially stronger on consumer electronics appliances. The reason for this weakness is that PCs are open and programmable, whereas Consumer electronics appliances are more tamper-resistant than PCs and therefore somewhat more difficult to hack the DRM application. Therefore, the dishonest user will find it much harder to bypass a consumer electronic (CE) device (the Client) audit logging mechanism. The audit logging mechanism is able to record all the user's actions, including her attempts to tamper with the audit logs.

The remainder of this paper is divided into sections as follows: Section 7.2 describes related work. Section 7.3 explains SK using the iButton. Section 7.4 discusses concisely the refinement we have made of SK. Section 7.5 gives our performance analysis on the implementation. Finally, section 7.6 concludes this paper also mentioning future work.

7.2 Related Work

Shapiro and Vingralek [2001] survey several mechanisms to manage the persistent state in a DRM system, including protected digital content, audit trail, content usage counts and decryption keys. One of the mechanisms they mention is secure audit logging. The two secure audit logging methods they cite are Bellare and Yee as well as Schneier and Kelsey.

Bellare and Yee [1997] (BY) propose a scheme to construct audit logs which possess the forward integrity property: The keys are altered on a regular basis by feeding different secret values to a family of pseudo-random functions that generate the message authentication code (MAC) over the entire log entries. If an adversary is able to compromise the current MAC key, it is unfeasible for her to deceive the historical entries generated because she is not able to fabricate the MAC keys for previous log entries. "Forward integrity" can be viewed as an integrity analogue of "forward secrecy" [Menezes et al., 2001]. A protocol possesses the forward integrity property if a compromise of long-term keys does not compromise the past keys. BY maintains the audit logs on untrusted machines.

Schneier and Kelsey [1998] (SK) uses a linear hash chain [Lamport, 1981] to link the entire audit log entries so that some forms of tampering can be detected. The hash chain is constructed by hashing each previous hash value of each log entry, concatenating with some other values. SK provides a complete secure audit logging protocol, from the log creation to log verification and viewing. A trusted machine is needed during log file creation but not at every log entry creation. The untrusted machine needs to communicate with the trusted machine from time to time to create new log file and to validate log files. Similar to BY, SK shares the "forward integrity" property, but SK uses a collusion-resistant hash function to generate the keys for MACs generation of each log entry.

SK and BY share a security weakness. If an adversary is able to compromise the untrusted machine at time t, i.e. obtains the key at time t, she is able to forge the log entry at time t. SK and BY are able to make illicit deletion of audit logs detectable. However, they are not able to *prevent* unauthorized removal of complete audit logs. Both SK and BY reckon that the deletion of log entries cannot be prevented by using cryptographic

methods, but only by using write-only hardware such as CD-ROM, or paper printout.

7.3 The Protocols

Figure 7.1 illustrates the protocols in our secure audit logging method. SK1 and SK2 are from SK. The others, P1 and P2 are our own protocols.

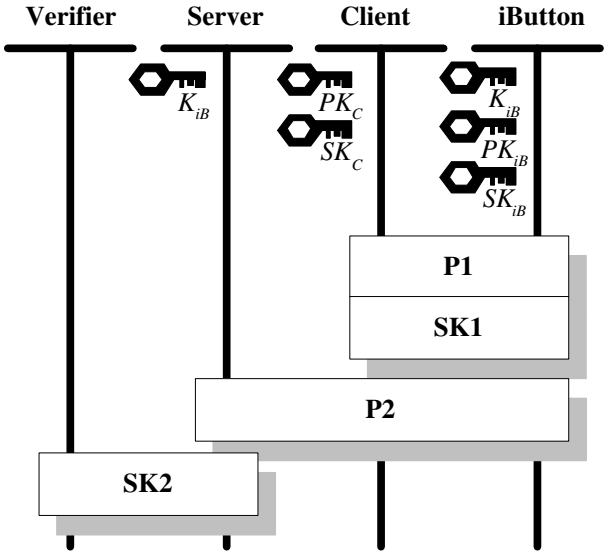

Figure 7.1: Overview of the secure audit logging method.

SK1 creates and closes an audit log. We focus on the technical details of the protocol and refer the reader to Reference [Schneier and Kelsey, 1998] for the motivations behind the protocol and other details. We use the notations listed in Reference [Schneier and Kelsey, 1998] to describe our protocols.

SK2 verifies and displays the audit logs to the Verifier. We have changed slightly the SK Verifier in that it does not store the Client's log file locally. The Verifier reads and verifies the log file remotely from the Server. In other words, the log file is stored securely in the Server, and the cryptographic processes are operated at the Server. Similar to SK1, Reference [Schneier and Kelsey, 1998] elaborates this protocol.

The Client and the iButton own different sets of key pairs. PK_C and SK_C are the public key and private key of the Client, respectively. PK_{iB} and SK_{iB} are the public key and private key of the iButton, respectively. The iButton and the Server share a secret key, K_{iB}. The public/private keys and the shared secret key are preloaded on the iButton before it is deployed.

We have made some assumptions that we believe to be reasonable while implementing the protocols. The main reason is to facilitate the implementation of SK in the resource constrained iButton.

1. Without restricting generality, there is only *one* Verifier, *one* Server, *one* Client and *one* iButton in the audit logging process. There is only *one* audit log file created from the process.

2. If the iButton is removed from the iButton reader halfway through an instruction then the log file is closed abnormally with a reason stated in the log.

7.3.1 P1

P1 generates the authentication keys, A_j and to generate the timestamp, d_j for recording the log entry. We have refined SK1 by deciding that the iButton should generate the keys and timestamps, as can be seen in Figure 7.2.

We improve SK1 to the extent that A_0, which represents the core security of SK1, and the timestamps are generated in a trusted subdomain. A fresh nonce, $Nonce$ is generated and stored on the iButton. The purpose of $Nonce$ is to ensure the freshness of the initial authentication key. The generated nonce is concatenated with the key, K_{iB}. Thereby, the initial authentication key cannot be generated by a malicious Client because she does not know the $Nonce$ and K_{iB}.

We use the iButton real-time clock to generate the timestamps. We encrypt the timestamps generated using the key, K_{iB} shared between the iButton and the Server. By doing so, the timestamps cannot be manufactured by the Client (who does not have access to K_{iB}).

The Client first requests an encrypted timestamp and an authentication key from the iButton for the current log entry. The iButton gener-

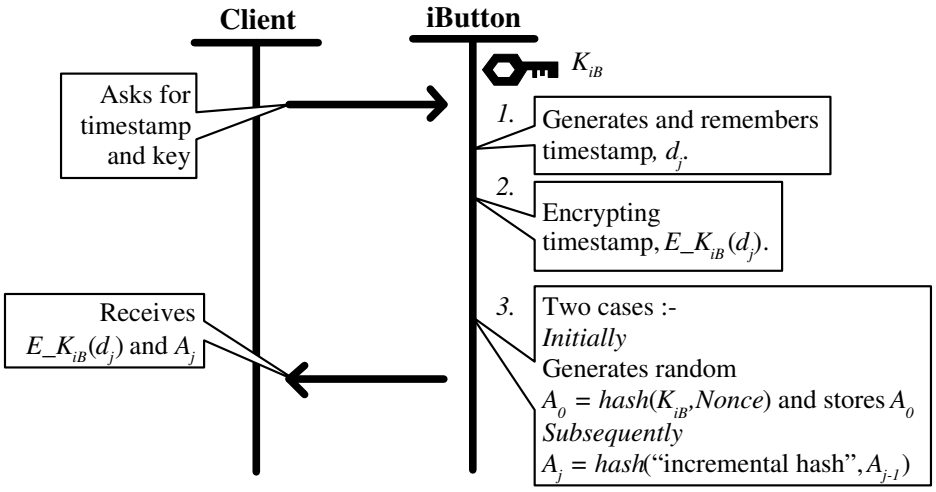

Figure 7.2: The P1 protocol for generating the initial authentication key A_0 and timestamp d from the iButton.

ates the timestamp using its real-time clock and encrypts the time stamp with the iButton secret key, K_{iB}. The iButton then remembers the first timestamp, i.e. the timestamp for the initialisation log entry, with type $"LogFileInitializationType"$ and also the last timestamp, for the close log entry, with type $"NormalCloseType"$ or type $"AbnormalCloseType"$. We will come back to this in section 7.4.

The iButton then generates a random key, as the initial authentication key, A_0 if it is the first log entry the Client constructs; otherwise, the iButton hashes the previously existing authentication key, A_{j-1} concatenated with a message to generate the next authentication key, A_j. The iButton stores the initial authentication key, A_0 and current authentication key, A_j for generating subsequent keys, A_{j+1}. After finishing the generations, the iButton sends the encrypted timestamp and the authentication key back to the Client.

7.3.2 P2

P2 synchronizes the iButton clock to the Server clock, which is a trusted clock. P2 also sends the log file maintained at the Client and the corre-

sponding initial authentication key on the iButton to the Server, as shown in Figure 7.3. The Server and the iButton share a secret symmetric key, K_{iB}, which is used to encrypt the data exchanges between the iButton and the Server (and also the timestamps) from the Client.

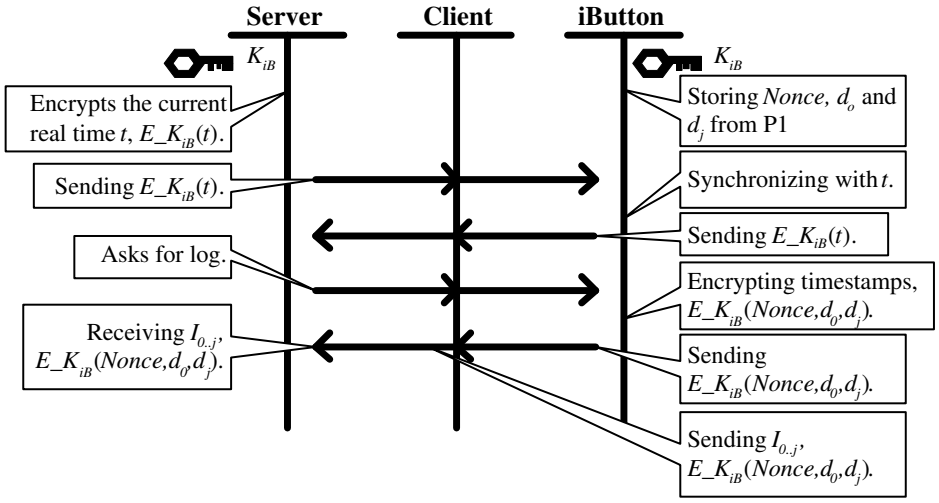

Figure 7.3: The P2 protocol of synchronizing the iButton real-time clock and sending audit logs to the Server.

The Server encrypts its current time with K_{iB}, and sends the encrypted time to the iButton for time synchronization. The iButton then decrypts the message and adjusts its real time according to the received time. Once the time is synchronized, the iButton sends back the accepted encrypted time to the Server as an indication that the time is synchronized.

After the clock synchronization, the Server sends a request message to the Client asking for the available log file. The Client forwards the request to the iButton. The iButton sends the encrypted $Nonce$ (the secret used to generate the initial authentication key), as well as initialisation timestamp, d_0 and close timestamp, d_j to the Server via the Client. At the same time, the Client sends the available log file (corresponds to the ID_{log} of $Nonce$) to the Server. In the DRM system, P2 is transparent to the Client.

7.4 SK Refinement

We have refined SK by introducing two auxiliary protocols, P1 and P2. P1 lets the trusted iButton instead of the untrusted client generate the authentication keys and the timestamps. The iButton remembers the timestamps for initialization and closing of the log. Additionally, the Client only possesses the encrypted timestamp from the iButton for audit logging, i.e. the integrity and confidentiality of the timestamps are ensured. P2 enables the iButton to transfer the encrypted initial authentication key and stored timestamps using the shared secret key with the Server. In other words, P2 is able to guarantee the integrity of the initial authentication key and the timestamps during the transmitting process.

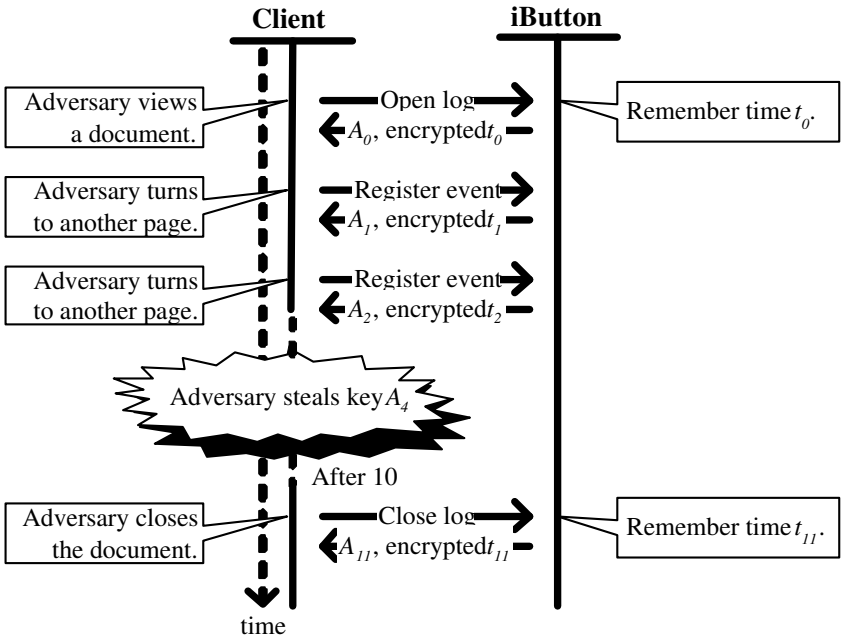

Figure 7.4: An adversary views a protected document and steals the key at time $t = 4$, during the logging process.

As pointed out by Schneier and Kelsey [1998, 1999], there is a security weakness in SK. If an adversary is able to compromise the Client by getting hold of the key A_t, directly after it has been generated at time t, the

adversary is able to falsify the log entry at time t. The adversary is also able to create more counterfeit log entries and remove some log entries. The Verifier is not able to detect the frauds because the adversary is able to construct another "truthful" hash chain and MAC over the entire log entries using the compromised authentication key.

In our refinement of SK, by using encrypted timestamps, we are able to detect some of the aforementioned frauds. If the adversary wishes to create more log entries, she has to obtain the cooperation of the iButton to generate valid timestamps. The adversary does not have the right key, so she cannot fabricate arbitrary timestamps herself. The adversary can reuse genuine encrypted timestamps, but the verifier will notice missing or duplicate time stamps, or time stamps that are presented out of order. The adversary will also be caught truncating the log file when the time of truncation does not match the time of the last transaction remembered by the iButton. In case of tampering, the user can be held responsible for the entire period between the log initialization time and close time.

We now present a concrete example of the difference between the original SK and our version as shown in Figure 7.4. An adversary, who owns a protected document and an associated license, starts the content renderer on the Client. The first log entry, initialization log, is then generated. Subsequent log entries are generated when she turns to other pages of the document. At time t_4, the adversary successfully steals the key A_4. She does not stop browsing the document, but keeps reading until time t_{10}. She closes the document at time t_{11} and so the log file is closed. She wants to deny the fact that she has viewed the document from t_4 till t_{10}. Instead the adversary wants the Verifier to believe that she has viewed the document until t_3. The adversary would wish to do so for example when she is charged on a pay-per-view basis. The adversary thus removes the log entries from t_4 onwards, and creates a false close log entry using timestamp t_4. As she possesses the key A_4, she is able to construct a valid hash chain and forge the MACs for this fraudulent log. In our version of SK, the Verifier is able to detect the forgery because the iButton remembers the log closing time (t_{11}), which in the scenario above does not match t_4. The original SK protocol does not detect this situation.

Suppose that the Client cheats by using key A_4 instead of A_{11} to close the log file. The next time the client connects to the server to get new

content she will need the cooperation of the iButton to authenticate herself. This gives the iButton the chance to present the latest key A_{11} to the server, and so the cheating Client will be found out.

Using encrypted time stamps improves the security of the protocol but weaknesses remain. For example if after a perfectly legitimate run of the protocols the user starts viewing the content using an application that does not log any actions, then this will not be noticed.

7.5 Performance Analysis

We are interested to know how the iButton affects the performance of the Client during audit logging. This allows us to determine if the iButton is practical for secure audit logging in a DRM system. We have run several performance tests on the implementation. We have used a PC machine Pentium III 800MHz with 256MB RAM as the Client machine for performing the tests.

We have measured the time for creating different numbers of log entries, from 1 to 10 on the Client. The graph is nearly a straight line. Generating 1 log entry takes approximately 1 minute.To explore why it takes roughly 1 minute to generate only 1 log entry, we have measured the time spent for performing cryptographic operations on the iButton. We believe that the cryptographic operations are the main causes to the long time taken for generating log entries.

We read the start time on the iButton right before the iButton starts the calculation under investigation. We read the stop time on the iButton once it stops the process. The time taken is the value of deducting the start time from the stop time. The result is then transmitted back to the Client. As only time with seconds-precision is available on the iButton (the software does not provide access to the $\frac{1}{256}$ second accuracy of the hardware clock), we have run the process repeatedly on the iButton, dividing the time measures by the number of repetition. We take the average value of 20 measurements as our final value using the standard deviation as error margin.

We evaluate the time spent for encrypting/decrypting a message of various size in bytes (from 8 to 128) using the 64-bit key DES algorithm on

the iButton. We realize that the times are around 200 ms for large message (size bigger than 128 bytes). We measure the time spent for hashing a message of sizes range from 8 to 128 bytes using the SHA1 message digest algorithm. The time spent for hashing 56 bytes is almost double the time spent for hashing 48 bytes. This is due to the message padding to 64 bytes [NIST, 1995]. We also measure the time consumed for encrypting a message of sizes from 8 bytes to 128 bytes, using 128-bit public key of RSA algorithm; and decrypt using the corresponding private key on the iButton. The RSA encryption takes averagely 25 seconds, while 22 seconds are needed for RSA decryption. We measure the time needed to sign a message using SHA1 and RSA and to verify the signed message. It takes 4 to 5 seconds to sign a message on the iButton, but it takes 5 to 6 seconds to verify the signature.

We do a back of the envelope calculation to confirm the measurement on the log entry generation time we obtain. The iButton takes less than 1 second for generating the timestamp and authentication key, i.e. to complete the protocol P1. The cryptographic operations on the iButton, as mentioned earlier consume most time. RSA private key decryption and public key encryption take approximately 24 seconds and 20 seconds, respectively. DES encryption and decryption need about 0.1 second, respectively. Signature generation and verification require roughly 5 seconds and 4 seconds respectively. These values are taken according to the size of the message we have used.

To conclude our performance analysis, as it takes about 1 minute just to generate 1 log entry, the iButton is not practical to be used in a system that requires frequent logging. However, if the system only logs the main events, such as playing a 4-minute song, reading an eBook, and other content access taking a longer time, we believe that the iButton is practical. Note that in our system logging overlaps with the actual content rendering.

For logging frequent events, we believe that we could use iButton as a bootstrap device for ensuring the trustworthiness of the first audit log entry, and we could do the subsequent log entries creation for frequent events without the presence of the iButton. This issue remains open for future investigation.

7.6 Conclusions and Future Work

We propose using secure audit logging in a DRM system where the Client is not permanently online. This allows the Server to obtain knowledge of the Client's behaviour during offline periods. We implement the Schneier and Kelsey (SK) secure audit logging protocol, using tamper-resistant hardware (an iButton) as the SK trusted machine.

The performance evaluation reveals that it takes about 1 minute for generating 1 log entry. We reckon that this is not practical for a system that requires frequent logging but feasible for a system that only needs to log the main events such as playing a 4-minute song. To make the iButton implementation also practical for recording more frequent events in future work we intend to use the current implementation recursively: one entire log on the untrusted PC would then correspond to one log entry that involves the iButton. The performance could also be improved dramatically using a more powerful iButton.

The main problem with all secure audit logging protocols is that if a log entry at time t is compromised, then none of the log entries after time t can be trusted. Our use of securely encrypted time stamps can solve this problem in some (but not all) cases. We believe that we can improve the security further by asking the iButton to do a little more work. This remains open for future work.

CHAPTER 8

LICENSE PROTECTION

License Protection with a Tamper-Resistant Hardware Token[7]

Cheun Ngen Chong, Bin Ren, Jeroen Doumen, Sandro Etalle, Pieter Hartel, and Ricardo Corin

Abstract Content protection mechanisms are intended to enforce the usage rights on the content. These usage rights are carried by a *license*. Sometimes, a license even carries the key that is used to unlock the protected content. Unfortunately, license protection is difficult, yet it is important for digital rights management (DRM). Not many license protection schemes are available, and most if not all are proprietary. In this paper, we present a license protection scheme, which exploits tamper-resistant cryptographic hardware. The confidentiality and integrity of the license or parts thereof can be assured with our protection scheme. In addition, the keys to unlock the protected content are always protected and stored securely as part of the license. We verify secrecy and authentication aspects

[7]This chapter has been published in 5th Workshop on Information Security Applications (WISA 2004), volume 3325 of LNCS, 2004, pages 224–238, Springer-Verlag.

of one of our protocols. We implement the scheme in a prototype to assess the performance.

8.1 Introduction

In a digital rights management (DRM) system, we use a *license* to specify the rights of a user on digital content [Chong et al., 2002]. For example, a commercial software license could restrict the execution of the licensed software to a particular number of uses. We must ensure the integrity of this information, so that the usage rights can be enforced correctly.

A license often carries the key to unlock the protected content. Therefore, we must ensure the confidentiality of this key, so that a dishonest user cannot access the content without abiding by the license. Additionally, the license can also carry *metadata* of the content, which may be as valuable as the content itself because metadata is critical for efficient content management. For example, the URI of the film. To ensure the availability of the film, the integrity of the URI must be protected.

Sometimes, we must ensure the confidentiality of some license information so that it cannot be accessed by any unauthorized users. For instance, a bank but not a content distributor can access a user's payment information specified on a license, e.g. credit card number. Therefore, the license, just as the content, requires adequate protection.

Unfortunately, license protection does not attract as much attention as content protection. There are only a few license protection schemes available, and most if not all are proprietary. Our main security objectives are to ensure confidentiality and integrity of a license or parts thereof, so that keys and metadata can be protected.

In addition, we would like to enforce different usage rights on different parts of the content. For instance, a patient record contains sensitive information about a patient. We want to protect and share this information by using different keys. The doctor is issued all the keys to access the entire patient record, but the insurance agent is only issued the keys needed to access insurance related information. To protect the patient record from being misused, we need to protect these keys.

In this paper, we propose a license protection scheme using a key tree

and a tamper-resistant hardware token with which we are able to achieve the aforementioned objectives. A hardware token provides a tamper-resistant environment for storage and for cryptographic operations; while a key tree grants us the flexibility to protect and share a license and content.

Our contributions can be listed as follows:

1. We propose a license protection scheme using a key tree and a hardware token, which is able to protect the license or parts thereof and content parts.

2. We perform an analysis of our protection scheme to justify its security properties.

3. We implement and evaluate the license protection scheme by using some off-the-shelf software tools and a Java iButton.

We have applied our license protection scheme according to a number of usage scenarios. To explain our approach, we choose a scenario of stock price, where a provider (e.g. NYSE) issues a license that restricts access of brokers (i.e., paid subscribers) and normal users to specific information on stock prices.

The organization of the remainder of the paper is as follows: Section 8.2 lists the security requirements. Section 8.3 briefly explains LicenseScript. Section 8.4 discusses our license protection scheme. Section 8.5 explains our prototype implementation. Section 8.6 reports on a performance evaluation of the prototype to justify the applicability. Section 8.7 briefly explains some related work. Finally, section 8.8 concludes and suggests future work.

8.2 Security Requirements

We assume that some of the system components can be trusted. This is more or less realistic with consumer electronic (CE) devices, but much less realistic when working on personal computers. In particular, we assume that the application interprets a license correctly. We treat this trusted part of the application as a *reference monitor* [Sandhu and Park, 2002]. For example, as soon as the license expires, the application stops

rendering. However, a malicious application can still cheat by tampering with the license. Therefore, we define the following requirements for our license protection scheme:

Requirement 1 License Integrity: The application must verify the integrity of the license when it accesses the license.

Requirement 2 Token Interaction: The application must interact with the hardware token to access the license and content parts.

Requirement 3 Key Confidentiality: The storage keys for accessing the license and content parts must be hidden from the application.

When these requirements are fulfilled, cheating by tampering with the license will be difficult.

8.3 LicenseScript License

In this section, we discuss our licensing language, LicenseScript. The language is based on multiset rewriting [Banâtre et al., 2001] and logic programming [Lloyd, 1987]. The reader may refer to our previous work [Chong et al., 2003a] for more detailed information.

A license has the following form:

```
license(Content,Clauses,Bindings)
```

Here, `Content` is (a link to) the content to be protected; `Clauses` is a Prolog program that decides if the operations performed are allowed or forbidden; and `Bindings` is a list of attributes that carry the status of the license and metadata of the content.

A clause has the following form:

```
Head :- Body_1,Body_2,...,Body_n.
```

Here, `Head` is the name and arguments of the clause, and the conjunction of `Body_1` up to `Body_n` is the body of the clause.

We use stock price scenario (as mentioned in section 8.1) for illustration. Figure 8.1 is an example of a LicenseScript license that allows a broker to view a stock price for 10 times. The license also allows the

provider to reset the number of times the stock price is viewed. Finally, the provider can update the stock price.

In Figure 8.1, `stock_price` is the link to the stock price; `get_value(X, Y, Z)` gets the value of the binding `Y` from the binding list `X` and unifies it with the variable `Z`; `set_value(V, X, Y, Z)` sets the value of the binding `X` from the binding list `V` with the value `Y` and stores the binding into the binding list `Z`; `is_member(X,Y)` checks if element `X` is a member of set `Y`; `get_curr_time(X)` gets the current time and stores it in `X` (primitives for other useful environmental data exist as well); `viewed` is the binding that stores the number of times the stock price has been viewed; `maxviews` stores the maximum number of times the stock price can be viewed; `updated` records the time that `stock_price` is updated; `subjects` is the access control list of subjects that can view `stock_price`.

In all clauses, `S` represents the subject making the query, `B1` is the current set of bindings and `B2` is the set of bindings resulting from a successful query. A failed query does not update the bindings. Clauses are triggered via external actions, for example if the `broker` presses the view button on the user interface, the `canview` clause is activated, with the appropriate settings for `S` (i.e., `broker`) and the bindings `B1` and `B2`.

8.4 License Protection Scheme

In this section, we introduce our license protection scheme. We use the architecture shown in Figure 8.3.

Four components are involved: the *application*, *reference monitor*, *token*, and *provider*. The application is a piece of software that interacts with the token, and which is used to access the license and the associated content. The reference monitor, which is a *trusted* part of the application, coordinates the actions of the application and the license. Each of these components has its own public/private key pair.

Two protocols support the communication between the components. Protocol A is used to send a protected license to the application from the provider. The provider generates the protected license and depending on its business model, decides which part of the license needs to be protected.

```
01)license(stock_price,
02)[(canreset(S,B1,B2):-
03)   S==provider,
04)   set_value(B1,viewed,0,B2)),
05) (canupdate(S,B1,B2) :-
06)   S==provider,
07)   get_curr_time(T),
08)   set_value(B1,updated,T,B2)),
09) (canview(S,B1,B2) :-
10)   get_value(B1,subjects,Ss),
11)   is_member(S,Ss),
12)   get_value(B1,viewed,X),
13)   get_value(B1,maxviews,Y),
14)   X <= Y, X = X + 1,
15)   set_value(B1,viewed,X,B2))],
16)[maxviews=10,
17) viewed=0,
18) updated=01012004,
19) subjects=[broker]])
```

Figure 8.1: A license that restricts a broker to access a stock price under 10 times.

```
license(stock_price,
[(canreset(S,B1,B2):-
   cipher("CJ...", skey1)),
 (canupdate(S,B1,B2) :-
   cipher("XY...", skey3)),
 (canview(S,B1,B2) :-
   cipher("AB...", skey4))],
[maxviews=cipher("12...",skey4),
 viewed=cipher("AC...",skey4),
 updated=01012004,
 skey1=cipher("89...",rootkey),
 skey2=cipher("aC...",rootkey),
 skey3=cipher("CC...",skey1),
 skey4=cipher("KL...",skey2),
 mac=cipher("XA...",rootkey),
 subjects=[(provider,rootkey),
           (broker,skey2),
           (alice,skey4)]])
```

Figure 8.2: Protected license of Figure 8.1, storing the storage keys and the MAC.

Protocol B is used when the application starts using the content, and when the reference monitor interprets the protected license. We will elaborate these protocols later in section 8.4.3. To use the license, the application must interact with the token and the reference monitor.

We will explain our scheme as follows: Firstly, we look at protected storage mechanisms, which have inspired our license protection scheme in section 8.4.1. Secondly, we show the structure of the protected license in section 8.4.2. Lastly, we illustrate the protection scheme protocols that we have developed in section 8.4.3.

8.4.1 Protected Storage Mechanisms

Our license protection scheme is a *protected storage mechanism*. Protected storage is defined by Pearson et al. [2003] as follows:

> Protected storage is a service to the host platform in which the trusted platform module (TPM) acts as a portal to confidential data stored on an arbitrary, unprotected storage media.

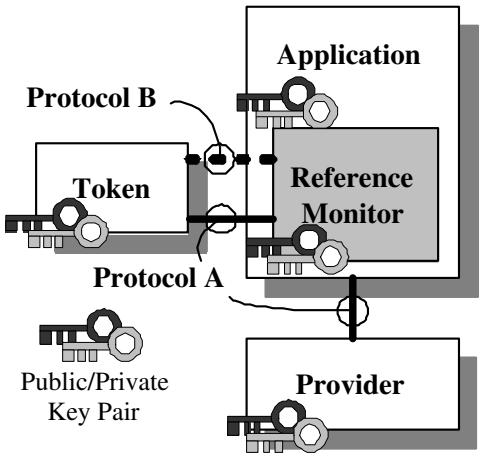

Figure 8.3: Overall license protection architecture.

Here, the TPM is a tamper-resistant cryptographic hardware module that is permanently embedded in a computer. The TPM can provide secure storage for keys and other sensitive information and it can perform cryptographic operations. Our license protection scheme uses an external hardware token instead of the TPM because this allows user the freedom to move licenses and content between machines.

In this paper, we use protected storage in the form of a *key tree*. This is a mechanism that has been used in secure group communication for key distribution and management [Goshi and Ladner, 2003].

Figure 8.4 provides an example of a key tree. A child node is encrypted using the storage key of the parent node. The root key is the "master key" for the whole tree. If say, `skey1` is needed to decrypt `data1`, the former will be decrypted using the `rootkey`. Then, `data1` can be decrypted with `skey1`.

For optimal performance, we use symmetric keys for the root key and the storage keys. The root key is stored on the token when it is issued and never leaves the token. It is sent to the user physically with the token. This root key is the secret key shared between the token and the provider. All decryptions take place on the hardware token for maximum security.

However, when sharing license information with another user, an actual storage key (which has become the root key for a sub-tree) must be

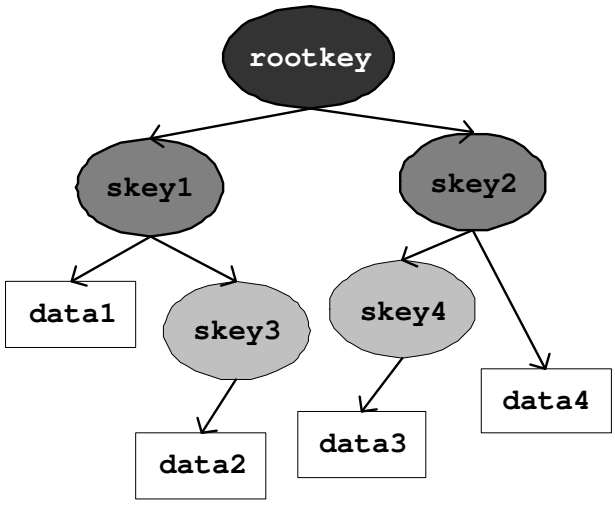

Figure 8.4: An example of key tree.

transferred to the user's token. For instance, we can allow a user to *only* access `data3` and `data4` by transferring the actual storage key `skey2` to the user's token. The process of transferring this storage key to another user's token falls outside the scope of this paper. However, we believe it can be achieved by using the TPM maintenance mechanism [Pearson et al., 2003], which is intended to transfer a storage key from a TPM to another TPM securely. In addition, we can exploit a secure transfer mechanism, such as the mechanism proposed by Atallah and Li [2003]. This deserves further study.

We can selectively deploy the information of the license with other entities by using the key tree. For instance, we can share the information of the license encrypted by `skey3` and hide the other information from another user, by using the key `skey2` as the root key for that user.

8.4.2 Protected License

By using a key tree, we can protect the license from Figure 8.1, as shown in Figure 8.2. Here, `cipher(X,Y)` is a predicate that stores the encrypted value `X` (which looks meaningless to human eyes) with the key `Y`.

Several additional bindings are needed to store the encrypted keys. For example, `skey1` is the encrypted key used to encrypt lines `03`–`04` of Figure 8.1, shown as `"CJ..."` in line `03` of Figure 8.2. The length of `"CJ..."` is the same as the original text shown in lines `03`–`04`. We use the binding `mac` (line `15`) to store the message authentication code, i.e., `"XA..."` of the license, which can be verified by using the `rootkey`. The `mac` can ensure the integrity of the entire license.

The `provider` has the `rootkey` (on her token). The `provider` can access all the encrypted data in the license. Therefore, the `provider` can execute clause `canreset`, which resets the value of the binding `viewed`; clause `canview`, which views the content of `stock_price`; and clause `canupdate`, which updates the content of `stock_price`. On the other hand, the `broker` can only execute the clause `canview`, because her token only has the actual key `skey2`.

In addition, we use the keys in this license to protect some information (i.e., parts) of the `stock_price`. The license only allows an authorized user (with the correct key) to access these protected parts. In Figure 8.2, `broker`, who is a paid subscriber, can access the `stock_price` information that is encrypted with key `skey2` and `skey4`. The user `alice`, who is not a paid subscriber can access less information, which is encrypted with key `skey4`, of the `stock_price`.

8.4.3 Protocols

In this section, we describe the protocols of our protection scheme: Protocol A (for transmitting the license) and Protocol B (for using the license). For the reader's convenience, we list the notation we use to describe the protocols in Table 8.1.

Protocol A (Figure 8.5) requires interaction between the hardware token, the application, and the license provider. Its two main objectives are : (1) to send the protected license to the application; and (2) to send the public key of the application to the token. The application's public key will certify the trustworthiness of the application when the license is used.

Symbol	Meaning
A	Application
R	Reference Monitor (the trusted part of A)
T	Hardware token
P	Provider
D	Data, i.e., license clause/binding or content
$\{X, Y\}$	Concatenation of X and Y
$\{\cdots\}_K$	Message is encrypted by key K
K_{st}	Storage key
K_{ss}	Session key
$K_{(X,Y)}$	Shared secret key of X and Y
K^+_{eX}, K^-_{eX}	Public and private key of X for encryption
K^+_{sX}, K^-_{sX}	Public and private key of X for signature
$S(M)_{K^-_{sX}}$	Signature of M with K^-_{sX}
$MAC(M, K)$	MAC of message M with K
Lic	License
Key	List of encrypted storage keys

Table 8.1: The notation.

$$
\begin{aligned}
&\text{A1. } A \rightarrow T \;:\; \{A, P, \text{“}name\text{”}\}\\
&\text{A2. } T \rightarrow A \;:\; \{N, MAC(N, K_{(P,T)}), T, A, P, \text{“}name\text{”}\}_{K^+_{eP}}\\
&\text{A3. } A \rightarrow P \;:\; \{A, \{N, MAC(N, K_{(P,T)}), T, A, P, \text{“}name\text{”}\}_{K^+_{eP}}\}\\
&\text{A4. } P \rightarrow A \;:\; \{Lic, \{N+1, A, K^+_{eA}\}_{K^+_{eT}}\}\\
&\text{A5. } A \rightarrow T \;:\; \{N+1, A, K^+_{eA}\}_{K^+_{eT}}
\end{aligned}
$$

Figure 8.5: Protocol A – The hardware token, the application and the license provider interact during the transmission of the protected license and the public key of the application.

A1 Application (A) asks Token (T) to get the desired license (identified by "$name$") from Provider (P). T must recognize P.

A2 T generates a fresh nonce N, a MAC of N (using the secret key shared with P), $K_{(P,T)}$, concatenates N, $MAC(N, K_{(P,T)})$ and identity T to the message received from A, encrypts the result with Provider's public key K^{+}_{eP}, and sends this to A. The fresh nonce is necessary to prevent a replay attack. The secrecy of the message can be assured by encrypting it with K^{+}_{eP}. The authenticity of the nonce (i.e., that it was produced by T) is guaranteed by $MAC(N, K_{(P,T)})$. Therefore, a malicious application cannot fabricate the message of step A2 without help of the token.

A3 A sends its identity and the received message from T to P.

A4 If P can decrypt the received message, P is implicitly authenticated by T. P increments N, concatenates $N + 1$ with A's public key K^{+}_{eA} and encrypts the result with T's public key K^{+}_{eT}. P sends this message and the protected license Lic to A.

A5 A forwards the encrypted message to T and stores Lic. The license can be stored securely because its content is protected by a key tree.

Protocol B (Figure 8.6) After Protocol A has finished, the application has the protected license. Each time the license is used, Protocol B is run.

As stated before, the reference monitor is assumed to be the trusted part of the application. We trust the reference monitor in the sense that it will correctly interpret each license. Also, we assume that the token has obtained, and trusts, the public key of the reference monitor initially.

The steps of Protocol B are:

B1 Application (A) wants to access the license. It initiates the interaction by sending to Token (T) its identity A, the license Lic and the MAC value of the license, $MAC(Lic, K_{(P,T)})$. If validation fails, T terminates the interaction and records the event.

B2 T verifies the integrity of Lic. If the integrity is violated, T terminates the interaction with A, and A cannot access the content.

$$
\begin{aligned}
\text{B1. } A \rightarrow T &: \{A, Lic, MAC(Lic, K_{(P,T)})\} \\
\text{B2. } T \rightarrow A &: \{K_{ss_1}\}_{K^+_{eA}} \\
\text{B3. } A \rightarrow T &: \{Key, \{D\}_{K_{st}}, \text{“}param\text{”}\}_{K_{ss_1}} \\
\text{B4. } T \rightarrow R &: \{\{Lic, D, S(D)_{K^-_{sT}}\}_{K_{ss_2}}, \{K_{ss_1}, K_{ss_2}\}_{K^+_{eR}}\} \\
\text{B5. } R \rightarrow A &: \{D\}_{K_{ss_1}} \\
\text{B6. } A \rightarrow R &: \{D'\}_{K_{ss_1}} \\
\text{B7. } R \rightarrow T &: \{D'\}_{K_{ss_2}} \\
\text{B8. } T \rightarrow A &: \{\{D'\}_{K_{st}}\}_{K_{ss_1}} \\
\text{B9. } A \rightarrow T &: \{Lic'\}_{K_{ss_1}} \\
\text{B10. } T \rightarrow A &: \{MAC(Lic', K_{(P,T)})\}_{K_{ss_1}}
\end{aligned}
$$

Figure 8.6: Protocol B – The application interacts with the token for using the license.

We use secure audit logging to record this incident [Chong et al., 2003e]. If the integrity is validated, T acknowledges A with a randomly generated fresh session key K_{ss}, encrypted with A's public key K^+_{eA}. Implicitly, the application is authenticated if the application can read this message using its private key.

B3 A retrieves a list of encrypted storage keys Key needed for the required data $D_{K_{st}}$, and sends it to T. The parameter “$param$” is used to identify the type of D, i.e., if D is a clause, “$param$” is the name of the clause. This message is encrypted with the session key K_{ss} to ensure the authenticity of the session.

B4 Before sending the decrypted data D, two cases are considered:

B4.1 If D is a license binding, T checks it against the previously stored value to assure that D has not been tampered with. If the check fails, the token terminates the transaction. Otherwise, T performs *application-specific* updates on the binding value stored on the token. For instance, the value of binding `played_times` is incremented. In any case, we log the bind-

ing values so that when T and A re-connect to P (Protocol A) say, for a new license or content, T sends P the stored binding values, so that P can check if the user has cheated [Chong et al., 2002]. If D is used for the first time, T will store it (T can trust the integrity of D at the *first time* because D is always encrypted with a storage key).

B4.2 If D is a license clause or a content part, no checking is done due to the limited resources of the token.

Then, T sends D and its signature $S(D)_{K_{sT}^-}$ as well as the license Lic encrypted with a new session key K_{ss_2} to the reference monitor R. The new session key is encrypted with the public key of R, i.e., K_{eR}^+.

B5 R verifies $S(D)_{K_{sT}^-}$ before interpreting the data D. Then, R sends D to A, encrypted with the session key K_{ss_1}. The encryption with K_{ss_1} is to ensure the authenticity of the session.

B6 After A has used and updated D (i.e., D' is generated), A sends D' to R.

B7 R checks if D' was updated correctly. If so, R sends D' to T encrypted with their shared session key K_{ss_2}.

B8 T replies to A with the encrypted D', i.e. $\{D'\}_{K_{st}}$. T encrypts it with the session key K_{ss_1} to ensure the authenticity of the session.

B9 A new license Lic' is re-constructed by A. A asks T to regenerate a new MAC value for the updated license Lic'.

B10 T sends the new $MAC(Lic', K_{(P,T)})$ to A to finish the final reconstruction of Lic'.

Steps B3 to B8 may be repeated in a session for different types of data (i.e., license clause, binding or content part) during the use of the license and content.

This completes the description of the protocols.

8.4.4 Formal Protocol Verification

We have used the protocol verifier CoProVe [Corin and Etalle, 2002] to verify Protocol A. Basically, what we needed to verify is that a malicious application would not be able to obtain the license without the correct intervention of the token. It is well-known that design of cryptographic protocols is rather error-prone, and that a great deal of published protocols has later been shown to contain errors prejudicing their safety. CoProVe helps at finding possible attacks and at proving that – under certain conditions – a protocol is attack-free. CoProVe works by taking as input a specification of the protocol and a system scenario describing the roles involved in the protocol – in our case token, application and provider of Protocol A – and by analysing all possible interleavings in presence of a malicious intruder.

We adopt two reasonable assumptions to keep our scenario simple yet expressive:

1. The token knows the genuine provider. It is a practical assumption because the token is dispatched by the provider.

2. The provider knows the genuine application. It is also practical because the provider keeps a list of authorized applications that can access the license.

Specifically, we have verified the following properties:

1. Secrecy: A fresh nonce must only be known by the token and the genuine provider. This is to prevent replay attacks.

2. Authentication: The malicious application and provider cannot impersonate the genuine ones. Thereby, the malicious application cannot impersonate a genuine one to decoy the token. We tested that a malicious application could not impersonate the token.

To carry out the verification we had to set up a finite-state scenario (consisting of a finite number of parallel sessions); this is the standard limitation of model-checking approach to verification: while we have checked scenarios with two parallel sessions (we are going to carry out tests with 3 parallel sessions as well) it is possible – though unlikely – that hidden flaws are revealed only by analyzing scenarios with a higher number

of parallel sessions. In the Appendix we report the source code used to check the secrecy of the nonce with two parallel sessions.

8.4.5 Security Analysis

In this section, we review the requirements of section 8.2 corresponding to our license protection scheme.

Requirement 1 is satisfied by using a message authentication code. The verification of the MAC value is performed on the token with the root key stored on the token (in Protocol B step B1). Therefore, we can ensure the correctness of the MAC verification because the secret key never leaves the token.

Requirement 2 is fulfilled *if* different parts of the content are encrypted by using different keys stored on the license (in Protocol B step B3). Therefore, the application must interact with the token *continuously* as long as the application accesses the content and license. Our protection scheme is aimed for content is short-lived, i.e., the value of the content is reduced after a short period of time. For instance, stock price. Therefore, once the content has been encrypted and presumably saved in the clear, we do not insist on communication with the token anymore.

Requirement 3 is satisfied. The keys stored on the license are encrypted. The decryption operations (on the keys, license clauses, bindings, and content parts) are performed on the token (in Protocol B step B4). However, during sharing, an actual storage key must be transferred from one token to another. This process is presumed secure by using some available mechanisms.

8.5 Prototype

In this section, we discuss a prototype implementation of our license protection scheme. The objective is to establish the applicability, and at the same time to conduct some performance evaluation. The prototype is built on a platform of Intel Pentium 4, 256 Mbytes of RAM, and a serial port connection with the iButton version 2.2.

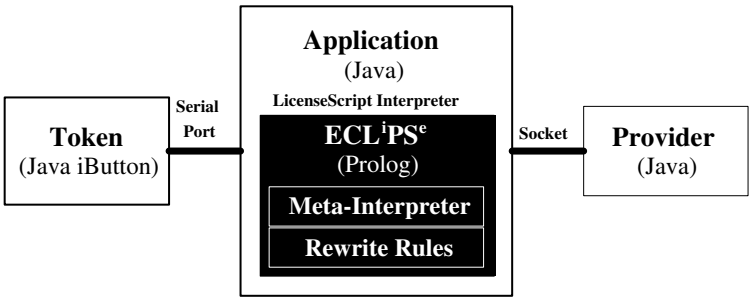

Figure 8.7: The architecture and components of the reference implementation.

We use off-the-shelf software tools to implement the components of our prototype, as shown in Figure 8.7:

1. LicenseScript License Interpreter, which is responsible for interpreting and calculating licenses. It *acts* as a reference monitor. We have used the LicenseScript Interpreter from our previous work [Chong et al., 2005] based on:

   (a) ECL^{i}PS^e: To execute the Prolog code retrieved from the LicenseScript licenses.

   (b) Meta-interpreter: To retrieve the clauses and binding values from the licenses and to send these to the ECL^{i}PS^e Prolog interpreter.

   (c) Rewrite Rules: To interpret the rights operations performed by the users via the application, for instance, play, copy, etc.

2. Application, which is used to access the license and the associated content, while interacting with the token. This is written in Java, using iB-IDE API, Java Cryptography Extension (JCE), and JavaCard Framework.

3. Token, which is a Java iButton version 2.2. It has a higher physical security than a normal smart card because the chip is physically protected by a stainless steel cover, and it supports common cryptographic algorithms.

4. Provider, which provides the protected license and sends it to the client via a socket connection. This is written in Java using JCE and Java.net.

After implementing the prototype, we performed several performance evaluations of the prototype.

8.6 Performance Evaluation

To verify the practicability of our license protection scheme, we perform several tests on our prototype.

From our previous experience, we know that the cryptographic operations on the iButton are slow [Chong et al., 2003e]. As it is used more frequently than Protocol A, we choose to evaluate the performance of Protocol B, in particular the two operations that involve the iButton: (1) decryption of keys and data (includes license clause, binding and content part) on different level of a key tree, on the iButton (section 8.6.1), and (2) reconstruction of the license, which involves encrypting the data and generating a message authentication code (section 8.6.2).

8.6.1 Test 1: Level of the Key Tree

The depth of a key tree influences the performance of our license protection scheme. For instance, to retrieve a license binding value, which is encrypted with a storage key at level 10 of the key tree, the token has to perform 10 steps of symmetric decryption (including the step to decrypt the encrypted binding value).

In this test, we measure the decryption time required at various levels, i.e., from level 2 to 10 inclusively (level 1 is the root key). The final result obtained for each level is the average of 5 repeated measurements. We run the test as shown in Figure 8.8.

The size of the data is less than 128 bytes. We found that it takes roughly 0.2 second for DES decryption (with a 56-bit key) on the iButton, which is consistent with the finding of our previous work [Chong et al., 2003e].

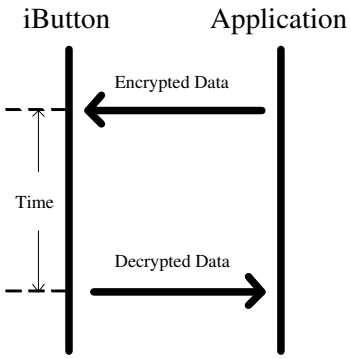

Figure 8.8: The procedure for measuring the time needed to perform data decryption at different levels of the key tree.

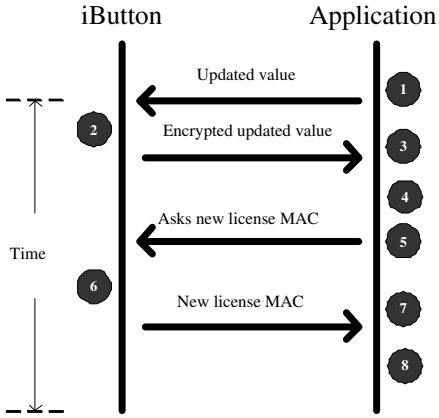

Figure 8.9: The procedure for measuring the time needed to perform data re-encryption on the token and reconstruction license on the application, at different levels on the key tree.

We derive a least square fitting (LSQ-Fit) formula to express our result of measurements:

$$t \;=\; 0.06 \pm 0.02 + (111 \pm 3) \times l \tag{8.1}$$

Here, t is the time in milliseconds required to perform DES decryption on the token for level l on the key tree. $l > 1$ because level 1 is the root key.

The first conclusion is that the depth of the key tree should be kept as low as possible. From Equation 8.1, it takes approximately 1.22 seconds to decrypt a data (of size less than 128 bytes) at level 10 of the key tree. This will cause a delay to the system, which is noticeable to the user.

8.6.2 Test 2: License Reconstruction

After the data is used and updated, we also need to re-encrypt the data on the token, reconstruct the license on the application and generate a new MAC for the updated license on the token.

We run a test, as shown in Figure 8.9. We use the same data size as less than 128 bytes to perform our tests, and we have run the same test for

the same data 5 times. The final result is the average value of these 5 tests.

We use an LSQ-fit formula to express our result:

$$t \;=\; 2256 \pm 80 + (2.56 \pm 0.28) \times l \tag{8.2}$$

Here, t is the time in milliseconds required to reconstruct the license for updated level l. The time required to reconstruct the license does not depend on the depth of the data in the key tree because only one DES encryption is performed on the iButton. Therefore, the time required to reconstruct the license for arbitrary updated level in the key tree is approximately 2.25 seconds.

We decompose the procedure of test into 8 parts, as shown in Figure 8.9, and test the time required for each part:

Parts 1, 3, 5, and 7 Application transmits less than 128 bytes of the data to the iButton and vice versa. We have run a test of the data transfer rate. It takes about 0.2 second to transmit less than 128 bytes of data, as shown in Figure 8.10. Our data size is about 100 bytes. Therefore, in total the communication between the iButton and the application takes **0.8 second**. Therefore, to update and reconstruct a license, it takes in total approximately 2.25 seconds, which is consistent with the overall measurement reported at the beginning of this section.

The graph shown in Figure 8.10 leaps drastically at around 120 bytes of data size. This is due to the iB-IDE API. When the data is over 120 bytes, it will be split into chunks for transfer, which causes more transmit time.

Part 2 This corresponds to Protocol B step B5. The iButton needs to perform a DES decryption with a 56-bit session key on the message to retrieve the updated value, which takes about 0.2 second [Chong et al., 2003e]. Then, the iButton encrypts the updated value with 56-bit storage key, and then with the 56-bit session key. Therefore, this process takes in total **0.6 second**.

Part 4 The application reconstructs the license with the encrypted and updated license data. This takes less than **0.05 second**.

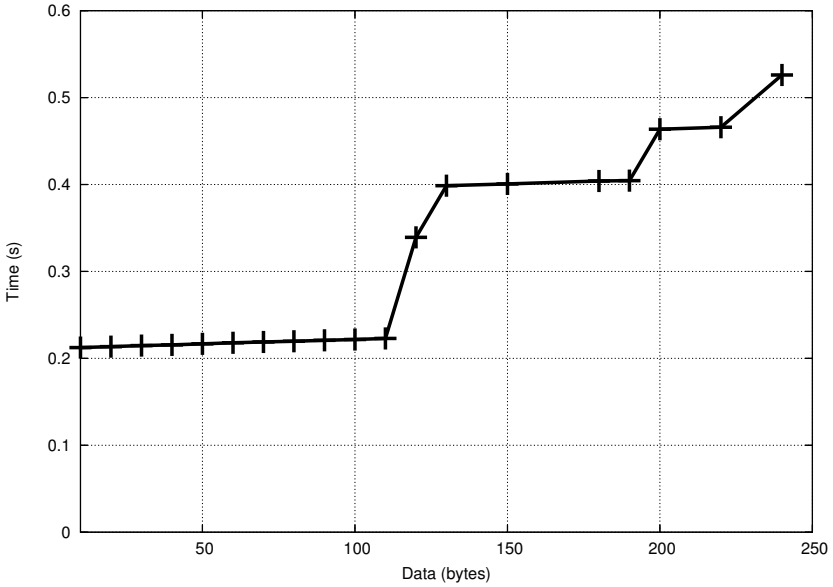

Figure 8.10: The data transmission time from the application to iButton.

Part 6 This corresponds to Protocol B step B7. Similar to step 2, it takes 0.2 second to decrypt an encrypted message with the session key. The iButton generates a MAC for the new license. The iButton needs about 0.15 second to generate a hash of the data size less than 128 bytes [Chong et al., 2003e]. The iButton needs 0.2 second to generate the MAC with the root key (DES encryption of the hash). Lastly, the iButton needs 0.2 second to encrypt the MAC with the session key. Therefore, this step takes in total **0.75 second**.

Part 8 The Application reconstructs the license by embedding the new MAC. Similar to 4, the application takes less than **0.05 second** to finish this final step of license reconstruction.

To conclude, the performance of the license protection scheme is acceptable from the user's perspective, if the data is small (less than 128 bytes). We may need a USB interface and a bigger token memory to handle bigger data. This remains for future investigation.

8.7 Related Work

In this section, we briefly discuss some related work. We investigate some XML documents security that XML-based rights expression languages exploit. We also discuss some commercial license protection mechanisms by using hardware tokens.

Damiani et al. [2000b] define and implement an authorization model for regulating access to XML documents. They exploit the capabilities of XML, and define an XML markup for a set of security elements describing the protection requirements of XML documents. Bertino et al. [1999] share the objective of Damiani et al but they focus on controlling the data access and dissemination of XML documents when there are XML documents exchanges between two parties. They discuss main protection requirements posed by XML documents and present a set of authorization and dissemination policies to achieve the aforementioned purpose.

As far as we are aware, the listed authorization models only propose the representation of the protected XML documents, e.g. new structure with new set of XML markups, etc. There is no protection operation mentioned how these protected XML documents are produced and accessed.

Several commercial proprietary protection schemes using hardware tokens are available. We are only able to scratch the surface of these mechanisms by studying their white papers. Most of them, e.g. Sospita (`http://www.sospita.com/`) and Wibu (`http://www.wibu.com/`) aim to protect a software code by executing parts of the code on their proprietary hardware tokens. They can lock this part of the software code unless the user pays for it. These protection schemes also assume that (parts of) the application that interfaces with the tokens are trusted.

Basically, protecting a license is only a secondary task in their scheme. Different from LicenseScript, their licenses do not have a rich structure to express complex usage scenarios. In addition, our license protection scheme, because of a key tree, allows flexibility in sharing a protected license and content with other users.

8.8 Conclusions and Future Work

A *license* is an important element of digital rights management (DRM) because it: (1) specifies a user's rights on a digital content, (2) carries a content key, and (3) describes metadata of the content. Therefore, we propose a license protection scheme based on a tamper-resistant hardware token and a key tree. The key tree provides flexibility and the hardware token provides tamper-resistance. We apply our license protection scheme to LicenseScript licenses. We analyze the protection scheme in terms of security with respect to some common security assumptions. We also perform a formal protocol verification using CoProVe.

We implement a prototype by using the Java iButton. To justify the practicability, we perform several measurement on the prototype. We conclude that the protection scheme is practical for a shallow key tree and small license size. We will extend our protection scheme for protecting fancy media, e.g. music or film. We will also use a USB connection for the iButton to improve the performance. Our scheme is intended for the business model of "one token to one provider" due to the limited resources of the token. However, we can extend our scheme to support "one token to many providers" – by using the public key of the token, we can generate a new shared secret key for a new provider. This remains as our future work.

Appendix: CoProVe for Protocol A

```
% Specification of Protocol A:

% specification for application role
application(A,T,P,N,L,K,[
    send([A,P]),
    recv([N, [N+K, [A, [P,T] ] ] ]*pk(P)),
    send([A, [N, [N+K, [A, [P,T] ] ] ]*pk(P)]),
    recv([L, [sha(N), [A,pk(A)] ]*pk(T)]),
    send([sha(N), [A,pk(A)] ]*pk(T))
]).

% specification of token role
token(A,T,P,N,K,[
    recv([A,P]),
    send([N, [N+K, [A, [P,T] ] ] ]*pk(P)),
    recv([sha(N), [A,pk(A)] ]*pk(T))
```

```
]).

% specification of the provider
provider(A,T,P,N,L,K,[
    recv([A, [N, [N+K, [A, [P,T] ] ] ]*pk(P)]),
    send([L, [sha(N), [A,pk(A)] ]*pk(T)])
]).

% secrecy check (singleton role)
secrecy(N, [ recv(N) ] ).

%scenario to check the secrecy of nonce with 2 sessions
scenario([[a1,Init1], %  application
         [a2,Init2], %  application
         [t1,Resp1], % token
         [t2,Resp2], % token
         [p1,Resp3], %  provider
         [p2,Resp4], %  provider
         [sec,Secr1] ]) :-
 application(a,_,p,_,_,_,Init1),
 application(m,_,p,_,_,_,Init2),
 token(_,t,p,n1,k,Resp1), % token knows genuine provider
 token(_,t,p,n2,k,Resp2),
 provider(_,t,p,_,l1,_,Resp3), % provider knows token
 provider(_,t,p,_,l2,_,Resp4),
 secrecy(n1, Secr1).

initial_intruder_knowledge([t,a,m,p]).
has_to_finish([sec]).
```

CHAPTER 9

STREAMING AUDIO PROTECTION

StreamTo: Streaming Content using a Tamper-Resistant Token

Jieyin Cheng, Cheun Ngen Chong, Jeroen Doumen, Sandro Etalle, Pieter Hartel, and Stefan Nikolaus

Abstract StreamTo uses tamper resistant hardware tokens to generate the key stream needed to decrypt encrypted streaming music. The combination of a hardware token and steaming media effectively brings tried and tested PayTV technology to the Internet. We provide a security analysis and present two prototype implementations with a performance assessment, showing that the system is both effective and efficient.

9.1 Introduction

To enforce usage rights and to prevent copyright violations, digital content needs to be protected. As shown in Figure 9.1, content protection has three objectives [Judge and Ammar, 2003]: (1) *protected distribution*, which protects content when it is accessed online by a content renderer,

e.g. streaming mechanism; (2) *protected storage*, which protects content while being stored locally, e.g. safe disc; and (3) *protected output*, which protects content after it is being rendered by a content renderer at a content output (e.g. a sound card), e.g. Microsoft Secure Audio Path (SAP).

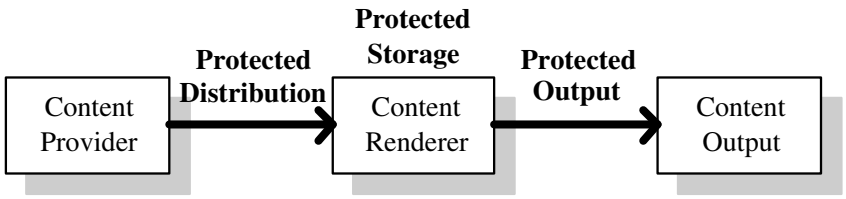

Figure 9.1: Three phases of content protection.

Content protection is difficult on a personal computer (PC) because most of the PC components (i.e., content renderer and content output) are open (i.e. programmable) and thus not trustworthy. When protected content is being used locally on a PC, an attacker might be able to retrieve the actual content by circumventing the protection mechanism [Greene, 2001].

However, if content is stored on a server while being used via a streaming mechanism (SM), the security of the content can be guaranteed to a certain extent because the *entire* content is not sent to the user's PC directly but piecemeal as a stream of packets [Holankar and Stamp, 2004]. This stream of packets is interpreted and rendered at the user's PC as they arrive. Therefore, SM helps to achieve *protected distribution* of content, provided that the stream cannot be captured easily.

Compared to a PC, a consumer electronic (CE) device is relatively more trustworthy because its components can be manufactured compliant and non- programmable [Eskicioglu and Delp, 2001]. Therefore, it is more difficult to circumvent the protection mechanisms applied to CE devices. A common example of such a content protection mechanism is the Conditional Access System (CAS) [Kravitz and Goldschlag, 1999]. A PayTV system [Jain et al., 2002] applies CAS to control users access to broadcast TV. Similar to SM, CAS is able to achieve *protected distribution* of the content.

In this paper, we propose StreamTo, which combines aspects of CAS and SM to design a content protection approach, supported by a tamper-

resistant hardware token, e.g. a USB dongle. We believe that in the foreseeable future, the hardware token will become common due to the convenience it provides to users and its reasonable price. A tamper-resistant hardware token can also provide *protected storage* for the content.

In addition, StreamTo has some additional benefits:

- It allows using content without an active Internet connection i.e., *offline*, or when the user does not have sufficient bandwidth. Similarly, the content provider does not have to worry about overloaded servers when there is a large number of users demanding online access.

- It allows flexible sharing of content between users. The provider can control access to different parts of the content by different users. This is useful for business-to-business (B2B) and business-to-consumer (B2C), for instance, when paying users can enjoy full content access, at near CD quality, while non-paying users can listen to clips only.

StreamTo is able to solve some of the security threats faced by CAS and SM. This will be discussed later in section 9.4. Like most streaming mechanisms, StreamTo is not easily scalable. Scalability could be achieved by using Broadcast Encryption (pioneered by [Fiat and Naor, 1994]). However, this is beyond the scope of the present paper.

Here, we show that StreamTo is applicable, practical and secure (within limits). To conclude, our contributions in this paper are:

- We propose StreamTo, which is able to provide a measure of protection for streaming content. The approach is inspired by concepts of CAS and SM. We analyze the security of the approach corresponding to common threats.

- We implement StreamTo on two commercial tokens, namely the CodeMeter Stick (CM-Stick) [Buchheit and Kügler, 2004] (`http://www.wibu.com`) and the Java iButton (`http://www.ibutton.com`). This shows the applicability of StreamTo.

- We assess the performance of the prototype to justify the practicality of StreamTo.

The remainder of the paper: Section 9.2 briefly explains CAS and SM, which inspired StreamTo. Section 9.3 describes StreamTo in detail. Section 9.4 analyzes the security of StreamTo, referring to common security threats. Section 9.5 implements a prototype on a CM-Stick and an iButton. Section 9.6 assesses the performance of the prototype. Section 9.7 discusses some related work. The last section concludes and presents future work.

9.2 CAS and SM

A Conditional Access System (CAS) is a smart-card-based technology [Guillou, 1984], which is used in PayTV systems. The smart-card stores subscription information and a secret key. A set-top-box (STB) is required to interface with the smart-card and the television (TV).

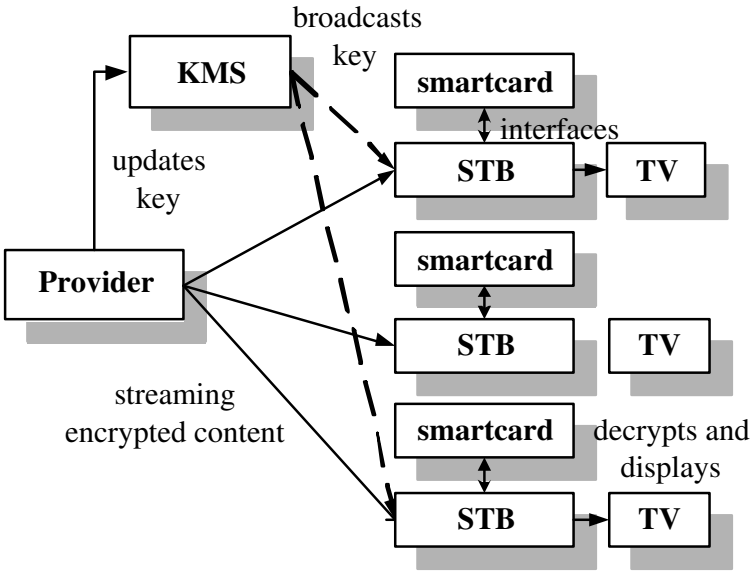

Figure 9.2: An abstract view of a conditional access system (CAS).

The provider encrypts a TV program using a content key (which is the same for all users) and broadcasts the encrypted TV program, as shown in Figure 9.2.

The key management system (KMS), which is responsible for billing, subscriber and key management, transmits the universal content key to the authorized subscribers. A content key is encrypted with the unique secret key stored on a smart-card [Macq and Quisquater, 1995]. The smart-card decrypts and stores the content key received from the provider (via STB). The STB decrypts the encrypted TV program with the content key, and displays the program on a TV.

The provider updates the content key used to encrypt the TV program/channel on a frequent basis (normally each 5 to 20 seconds). Once the key is updated, the KMS must retransmit the updated content key to the subscribers within seconds.

In a streaming mechanism (SM), as shown in Figure 9.3, the provider encrypts the content with different content keys for different users. The content is encrypted and transmitted to the user *piecemeal*, i.e., packet by packet.

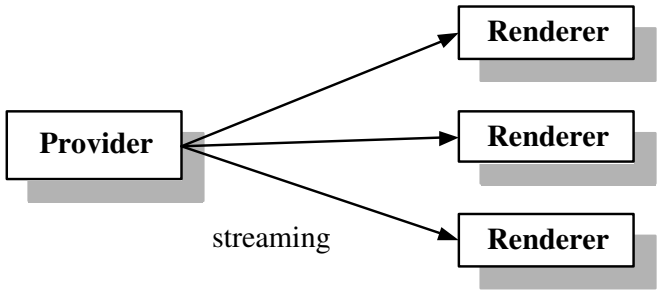

Figure 9.3: An abstract view of a streaming mechanism.

A user has a renderer, which is a software application that establishes a secure channel with the provider. The content key is transmitted to (or generated at) the renderer when a secure streaming session is established. The renderer then decrypts the content packet by packet with the content key and renders it, as it is received, leaving behind no residual copy of the content (packets) at the PC (assuming that the renderer is not hacked).

The characteristics of the content key of CAS, SM and StreamTo differ

	Uniqueness	Update
CAS	A content key is shared among all authorized users.	The content key is updated frequently.
SM	A unique content key is assigned to a user.	The content key is not updated in a streaming session.
StreamTo	A unique content key is assigned to a user.	The content key is updated frequently.

Table 9.1: Comparison of CAS, SM and StreamTo with respect to the characteristics of the content key.

as shown in Table 9.1. We list the two most important characteristics of a content key: (1) uniqueness (whether the key is unique for different content and user), and (2) update (whether the key is updated on a regular basis).

9.3 StreamTo

In this section, we discuss StreamTo, which is outlined in Figure 9.4.

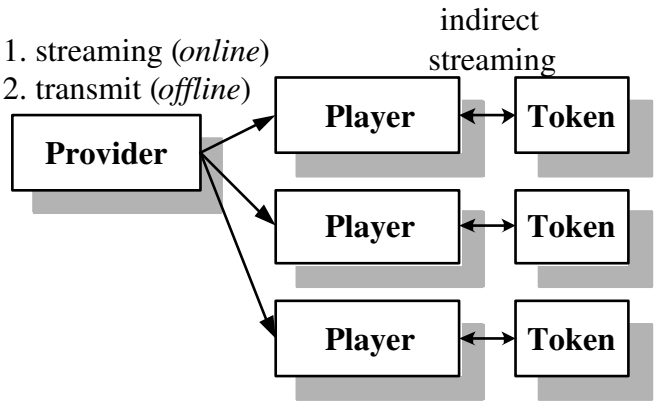

Figure 9.4: An abstract view of StreamTo.

We use a *token*, which has a cryptographic co-processor and tamper-resistant storage. The token is dispatched physically by the *provider* to a user in the same way as a PayTV smart-card. The provider also serves encrypted content. A user has a customized *player* (a software application) that interfaces with the token, and which can play encrypted content. The player depends on the token for providing the key stream necessary to decrypt the content stream.

StreamTo can handle two methods of rendering the content, as shown in Figure 9.4: *online* and *offline*. For online rendering, the provider streams the content to the player; whereas for offline rendering, the provider transmits the entire encrypted content to the player. For both access methods, the player plays the content *piecemeal*, waiting for each subsequent block of the key stream from the token, i.e., *indirect streaming*.

StreamTo has the characteristics of both the CAS and SM:

- The provider generates a unique content key for a user (SM provider).
- The player decrypts and plays the content piecemeal (SM renderer).
- The token stores a secret key (CAS smart-card).
- The content key is updated frequently (CAS provider).
- The token transmits the updated key for decryption to the player (CAS KMS and smart-card).

To explain the StreamTo protocols in detail, we use the notation listed in Table 9.2.

We explain the types of key that we use in StreamTo in section 9.3.1. We describe the encryption and decryption process in section 9.3.2 and section 9.3.3, respectively.

9.3.1 Keys

We use three different key types: secret key, content key and key stream.

- A secret key ($SecK$) is a secret shared between the provider and the token. We assume that an attacker cannot read, modify or access this

Notation	Meaning
$SecK$	A secret key shared between the token and the provider.
K_i	The content key for i^{th} content frame.
S_i	The key stream for i^{th} content frame.
P_i	The i^{th} frame of content (plaintext).
C_i	The corresponding i^{th} frame of encrypted content (ciphertext).

Table 9.2: The notation of the StreamTo protocols.

key stored on the token; it never leaves the token, and is preloaded to the token in a secure environment of the provider.

- A content key (K_i) is used for generating the key stream. The first content key K_0 is generated randomly by the provider and sent encrypted (with the secret key) to the user along with the encrypted content.

- A key stream (S_i) is used to en/decrypt the content. The key stream is derived from the content key and the content.

The size of the content key is short (e.g. 128 bits) so that a provider can send it to a player efficiently. The size of the key stream is equal to the size of the content so that stealing the key stream is inconvenient.

As a refinement, the provider could partition the content, using a different K_0 for each partition. This would allow for example free use of trailers but paid for use of the remaining content. We can also enforce different usage rights on different parts of content by using our LicenseScript license protection scheme [Chong et al., 2004].

In this paper, for simplicity, we only use one content key to explain StreamTo in the subsequent sections.

9.3.2 Encryption Process

Streaming content, e.g. an MPEG audio/video has a special structure: it is composed of multiple frames, each of which has a descriptive header. This

header contains the particular information for the corresponding frame, e.g. bit-rate, sample-rate, etc.

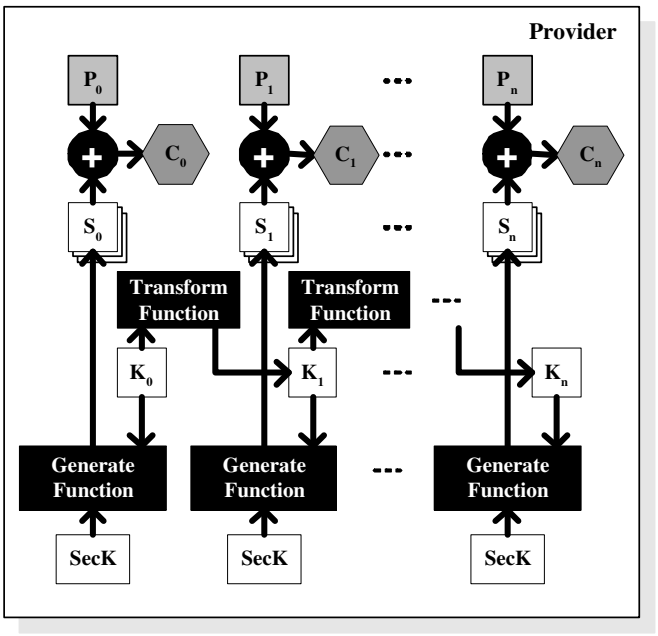

Figure 9.5: Encryption of streaming content frame by frame at the provider with a block of key stream, which is generated from a different content key.

StreamTo exploits this special feature of streaming content as follows:

$$S_i = \mathbf{generate}(K_i, SecK) \quad (9.1)$$

$$K_{i+1} = \mathbf{transform}(K_i) \quad (9.2)$$

$$C_i = P_i \oplus S_i \quad (9.3)$$

Here, i is an integer, $0 \leq i < \text{last\_frame\_number}$. The provider uses two functions: (1) a *generate function* (Equation 9.1), e.g. a pseudo-random number generator, which generates a key stream from a content key; and (2) a *transform function* (Equation 9.2), e.g. a keyed hash function, which updates a content key. These functions must be supported by the token to perform the decryption process, as will be discussed in section 9.3.3.

The encryption process, as shown in Figure 9.5 is performed by the provider. The provider generates a first content key K_0 randomly. The

generate function (Equation 9.1) takes the content key K_i and the secret key to produces a block of key stream for the current frame (P_i). The encrypted frame (C_i) is then XOR-ed with the block of key stream, as shown in Equation 9.3. Finally, the next content key is calculated by the transform function. If the output of the generate function is shorter than the frame size, it is repeated to form the required length.

We use the secret key $SecK$ in the generate function to ensure that the encrypted content is bound to the token. In addition, we want to assure that an attacker cannot derive a valid content key or key stream without the correct secret key.

The encrypted frames ($C_0, \cdots, C_n$) are written to a new content file, preceded by a header. The header contains the first content key (K_0) (encrypted with the secret key $SecK$ of the token), padding information and information about the generate and transform functions.

9.3.3 Decryption Process

The player receives an encrypted content file from the provider. When the player plays the encrypted content, the decryption process is executed as shown in Figure 9.6.

The player interprets the header information of the encrypted content to retrieve the encrypted first content key K_0 and other information. The player then asks for a valid token. The authentication can be achieved between the player and the token with standard methods [Kelsey and Schneier, 1999]. This falls outside the scope of this paper. The player then feed the token with the encrypted first content key (K_0), so that the token can decrypt it.

The token uses the generate function (Equation 9.1) to re-generate the key stream, and sends it to the player. The player retrieves a frame C_i from the encrypted content file, and decrypts it (by XOR-ing) with key stream S_i generated by the token, as shown in Equation 9.4.

$$P_i = C_i \oplus S_i \tag{9.4}$$

The player then updates the content key using the transform function (Equation 9.2), sends it to the token to generate the next key stream, and next frame is decrypted with this key stream.

At the *same time*, the player plays the previously decrypted frame P_{i-1}. The decrypted frames will be overwritten by newly decrypted frames after they are played (assuming the player has not been hacked).

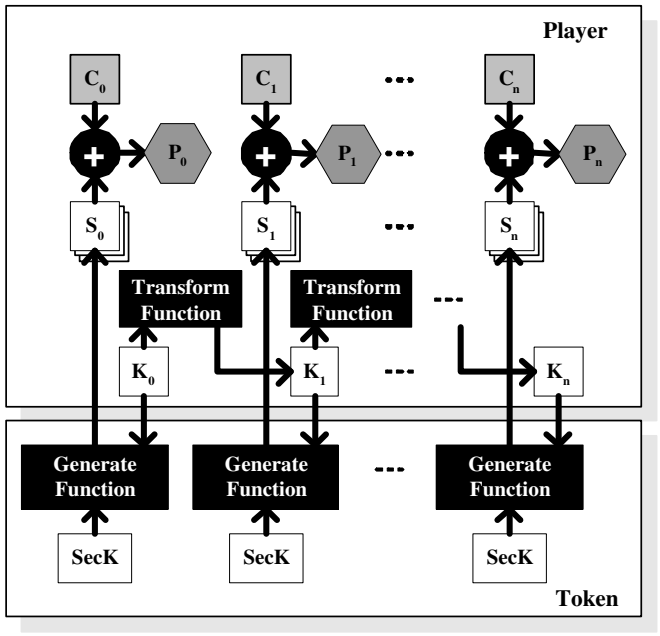

Figure 9.6: Decryption of encrypted streaming content frame by frame with the key stream using the content keys, while playing the decrypted frame.

9.4 Security Analysis

In this section, we analyze the security of StreamTo. We consider several known security threats: analog hole, digital hole, streaming theft, jugular attack, and cloning.

9.4.1 Analog and Digital Hole

The main security threat to PayTV and streaming is the widespread proliferation of content-capturing tools [Wald, 2002]. This threat is known as the "analog hole" [Holankar and Stamp, 2004], and is considered beyond

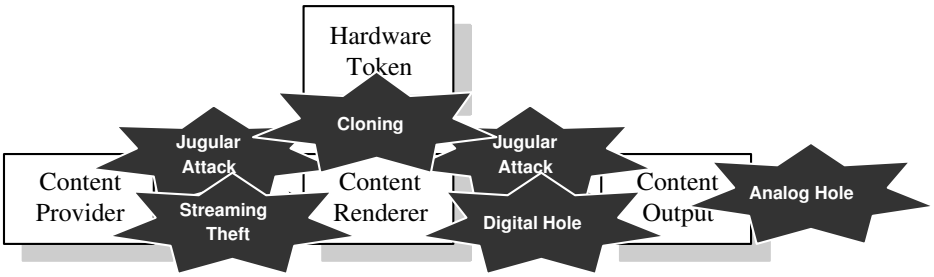

Figure 9.7: Common security threats to content protection approach.

the boundary of normal content protection approach. Therefore, StreamTo is not concerned with this security threat either.

The digital hole is another severe threat to common content protection approach, especially for the PC. The digital hole opens between the moment of decrypting and the moment of rendering the content. An attacker might be able to retrieve the plaintext content from the memory (especially on the PC) because the memory is always susceptible to manipulation and observation.

On a PC not much can be done to close the digital and analog holes; on CE device, the digital hole is more or less plugged, but the analog hole is still present. Watermarking is the only viable technique, which is able to provide some measure of protection [Haitsma and Kalker, 2002].

9.4.2 Streaming Theft

An attacker can steal streaming content simply by copying the source target, e.g. URL of the streaming content, and publish it on, say, another Web page on another server. Therefore, users can access the streaming content stored on the provider's server through the attacker's Web page.

In essence, the attacker is not only stealing the streaming content but is also stealing the original provider server's bandwidth. This can induce costs to the provider and can be difficult to track down.

In StreamTo, only authorized users, who possess tokens dispatched by the original provider, can access the streaming content. Therefore, it is not possible for other unauthorized users to access the streaming content through another server.

9.4.3 Jugular Attack

In a Pay-TV system, an attacker is able to obtain the plaintext keys by watching the output of the smart-card. The attacker can then redistribute these keys in near real-time to other users, allowing others to view the encrypted broadcast. The attacker can charge these users a cheaper fee. This attack exploits the vulnerability of using a universal content key for a TV program/channel for all users, and is known as the jugular attack [McCormac, 1996].

Online StreamTo is safe from this attack because we use different content keys for different users. Therefore, the attacker's keys are useless to other users, even if she is able to obtain the keys and redistribute them in near real-time.

However, for offline usage, a jugular attack poses a severe threat because the encrypted content file is stored at a user's PC assuming a hacked renderer. An attacker can obtain the keys by playing the encrypted content just once: XOR the ciphertext and the plaintext provides the key stream; or XOR the ciphertext and the key stream stolen provides the plaintext.

In any case, StreamTo is reasonably secure for short-lived content, i.e., if the value of he content is reduced after a short period of time. For instance, a breaking news video broadcast, a stock quote price, etc. Therefore, once the content has been decrypted and presumably saved in the clear, we do not insist on communication with the token anymore, i.e., the offline jugular attack and plaintext key stream do not pose a severe threat to StreamTo anymore.

9.4.4 Cloning

Cloning [Rao et al., 2002] is a physical security threat, from which PayTV systems suffer. A valid smart-card of a set-top-box can be cloned and distributed illegally. This is achieved for example by using side-channel attacks [Anderson and Kuhn, 1997] to steal and copy the secret key of a smart-card. Thereby, home users can purchase these smart-card clones for a lower price from an illegal distributor to access PayTV programs.

A hardware token provides a higher physical security than a smart-card because the chip is protected by a more physical means, e.g. stainless

steel casing. Additionally, it is more expensive to clone a hardware token than a smart-card.

In conclusion, we have argued that StreamTo is more secure than a typical smart-card-based PayTV system and a streaming system. Other content protection approaches have been proposed attempt to resolve some of the aforementioned security threats, which will be discussed briefly in section 9.7.

9.5 Prototype

In this section, we discuss the implementation of our prototype. We use streaming audio (MP3) in our prototype because it is less demanding on resources than video. If StreamTo can be applied practically to protect streaming audio, we can investigate if StreamTo can support other streaming content.

Section 9.5.1 presents an architectural overview of our prototype using the iButton and the CM-Stick, and introduces the software tools that we have used. Section 9.5.2 elaborates the implementation of StreamTo.

9.5.1 Architecture

The architectural overview of our prototype is given in Figure 9.8. The Provider and the Player are the two applications we have created. The Provider executes the encryption process discussed in section 9.3.2. It takes as input an MP3 audio file and produces an encrypted audio file as output. The Player, performs the decryption process discussed in section 9.3.3. It asks the token (i.e., CM-Stick or iButton) continuously for blocks of key stream to decrypt the audio.

We use five software development kits (SDK) to create the Provider and the Player for the CM-Stick and the iButton: (1) Windows Media Format (WMF) SDK (`http://msdn.microsoft.com/av/`), which supports various streaming audio standards, such as WMA and MP3; (2) CodeMeter (CM) SDK [WIBU, 2003], which supports interfacing with a CM-Stick; (3) Java SDK, which provides a standard cryptography library; (4)

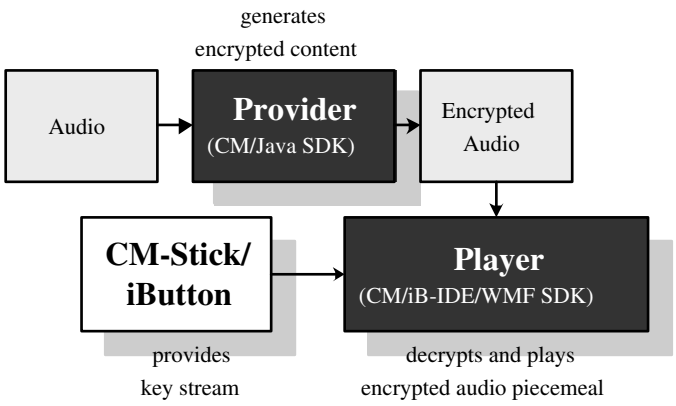

Figure 9.8: Architectural Overview of the prototype.

Javazoom, which supports MP3 audio; and (5) iB-IDE SDK, which supports interfacing with an iButton.

The hardware token we use in our prototype are a CodeMeter Stick(CM-Stick) and an iButton. Their features are illustrated in Table 9.3 (Note that the speed of the iButton is not constant and cannot be determined by external means [Kingpin, 2002]).

	CM-Stick	iButton
Manufacturer	WiBu-Systems AG, Germany	Dallas Semiconductor, America
Cryptographic coprocessor Speed	24 MHz	10–20 MHz
Non-volatile memory	128 kBytes	134 kBytes
Cryptographic algorithms	AES, Triple-DES (for communication), ECC, SHA-256	DES, Triple-DES, RSA and SHA-1
Interface	USB connection	Serial/Parallel and USB connection

Table 9.3: Comparison of the iButton and the CM-Stick.

9.5.2 Implementation

In this section, we give the implementation details of our prototype for StreamTo. We use the standard Counter-mode (CTR-mode) symmetric encryption [Lipmaa and Rogaway, 2000] to implement StreamTo by virtue of the simplicity, efficiency and proven security of CTR-mode encryption.

The content key (K_n) is the counter of CTR-mode encryption, which is initialized to a random n-bit string. The update function of the content key (Equation 9.2) is simple:

$$K_{i+1} = K_i + 1$$

The generation of the key stream (S_n) (Equation 9.1) is the encryption of the counter in CTR-mode encryption:

$$\begin{aligned} S_i &= \mathbf{AES\_CBC}(K_i, SecK) \quad (\mathrm{CM-Stick}) \\ S_i &= \mathbf{DES\_ECB}(K_i, SecK) \quad (\mathrm{iButton}) \end{aligned}$$

Here, $\mathbf{AES\_CBC}$ is AES encryption in CBC mode, which is the only symmetric encryption supported by the CM-Stick; whereas $\mathbf{DES\_ECB}$ is DES encryption in ECB mode, which is supported by the iButton.

In our prototypes, each time a new frame is decrypted a click is audible. This allows us to point out during demonstration when decryption happens.

In next section, we assess the performance of our prototype to justify the practicality of StreamTo.

9.6 Performance Assessment

To justify the practicality of StreamTo, we assess the performance of our prototype. Our prototype is built on a platform with an Intel Pentium 4, 1.4 GHz, 512 MBytes RAM, 20 GBytes hard disk space, running Windows XP. We use a 1-minute 192 kbps MP3 audio as the sample for our performance assessment. The sample has 2300 frames, each of which contains 623 bytes.

In our prototype, we use a serial connection for the iButton and a USB connection for the CM-Stick.

9.6.1 Content Key Size

The key stream is generated on the token by using the firmware symmetric encryption algorithm. Therefore, to determine if the content key size influences the performance of our prototype, we assess the performance of symmetric encryption of the iButton and the CM-Stick.

From our previous experience, we know that the cryptographic operations on the iButton are relatively slow [Chong et al., 2003e]. DES encryption (ECB mode) of 128 bytes on the iButton takes roughly 200 ms [Chong et al., 2004].

We also need to measure the time required by the CM-Stick to perform AES (CBC mode) encryption, which we use to generate the key stream. We use an LSQ-fit equation to summarize the result of 10 measurements as follows:

$$t \;=\; (0.5 \pm 0.002) \times d + (36 \pm 26)\ \text{ms}$$

Here, t is the time required in milliseconds and d is the data size in bytes. Thus, it takes approximately 100 ± 25 ms to encrypt 128-byte of data on the CM-Stick, making the CM-Stick about twice as fast as the iButton. This is consistent with the cryptographic co-processor speed (Table 9.3).

If we use 128 bytes of content key, i.e., 1024 bits, the iButton requires approximately $2300 \times 0.2 = 460$ seconds to generate the key stream, whereas the CM-Stick needs roughly 230 seconds. For a 1-minute MP3 this is too long, hence, we must sacrifice security for performance by (1) using a smaller content key size; and (2) en/decrypting every n-th frame of the audio sample only.

In our prototypes, we choose a content key of size 8 bytes (64 bits) for the iButton and 32 bytes (256 bits) for the CM-Stick. On the CM-Stick, it takes approximately 40 ± 26 ms to generate a block of key stream. However, for the iButton, it takes approximately 70 ms due to the slower co-processor. Therefore, we also use the second tradeoff on the iButton prototype, as will be discussed in next section.

9.6.2 Sample Bit Rate

The MP3 sample bit rate refers to the transfer bit rate for which an audio file is encoded, e.g. an MP3 file encoded at "at a bit rate of 128 kbps"

is compressed such that it can be streamed continuously through a transfer link providing a transfer rate of 128 thousand bits per second. The lower the bit rate, the more the audio file is compressed, and the worse the playback sound quality becomes.

On the other hand, the sampling frequency refers to the number of samples of an audio taken per unit time, i.e., the rate at which audio signals are sampled into digital form. Higher sampling frequency implies higher-quality of the digital audio.

The frame size varies with the sample bit rate and sampling frequency according to the MPEG-3 standard. We use 6 different sample bit rates (with the same sampling frequency of 44.1 KHz) of our experiment, as shown in Table 9.4. Each frame has standard constant time length of 26 ms. Therefore, the number of frames is determined by the time length of the entire MP3 audio. Therefore, for 1-minute MP3, it has approximately 2300 frames.

Sample Bit Rate (kbps)	File Size (MBytes)	Frame Size (bytes)
64	0.46	205
128	0.92	414
160	1.14	518
192	1.37	623
224	1.60	727
256	1.83	832

Table 9.4: Different sample bit rate, with different audio file size and average frame size.

As mentioned earlier, we use a serial port connection for the iButton prototype. From our earlier experience, the communication between the iButton and the player via serial connection causes a long delay [Chong et al., 2004]. For instance, transmitting up to 128 bytes from the iButton to the player requires approximately 0.2 second.

In addition, it takes roughly 80 ms to generate a block of key stream. Therefore, for decrypting the audio sample (192 kbps) of 2300 frames (1 minute of play time) by using content key of 64 bits, theoretically the iButton needs approximately $2300 \times (0.08 + 0.2 \times 2) = 1104$ seconds in total to generate and transmit the key stream to the player/

To overcome this deficiency, we modify our iButton prototype so that StreamTo only en/decrypts every n-th frame of the audio sample. Figure 9.9 shows the measurement for $n = 25,\ 50, \text{and } 100$. We take the average of 10 measurements. The decryption time measured includes the time required to upload the updated content key; to generate and transmit a block of the key stream; and XOR-ing of the encrypted frame. The graph $y = 60$ indicates the actual play time of the audio sample, i.e., 1 minute.

As can be seen in Figure 9.9, the graphs of the iButton are slightly slant, indicating that the time required to decrypt the audio frames increases with faster sample bit rate. It is simply caused by the preprocessing of the encrypted audio file: reading the audio frames from an encrypted audio file.

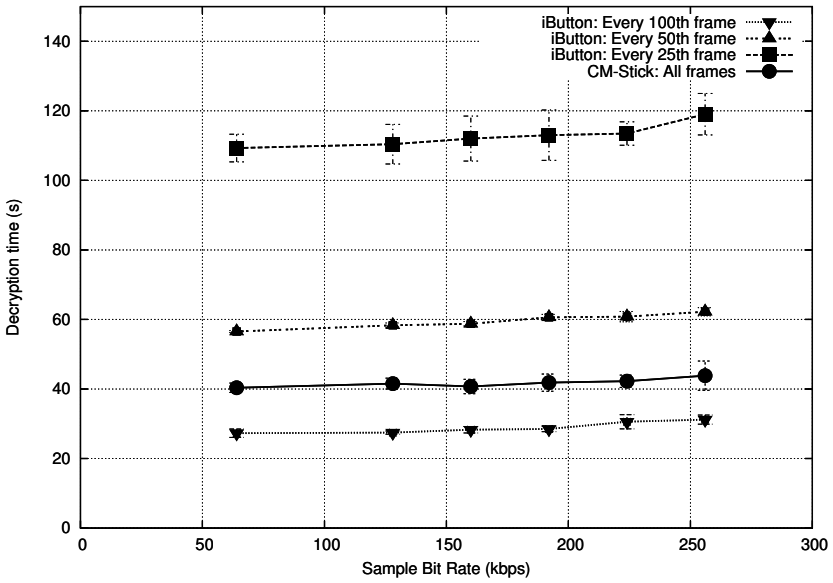

Figure 9.9: The time required to decrypt the encrypted audio sample frame by frame with the iButton and the CM-Stick.

When $n = 100$, the iButton StreamTo is able to handle the key stream generation and audio frames decryption more efficiently, compared to when $n = 25$. When $n = 50$, the decryption time is marginally parallel with the play time of the audio sample.

On the other hand, CM-Stick, due to its USB interface and faster cryptographic co-processor, has far better performance than the iButton with the serial interface, as shown in Figure 9.9. The time required to decrypt all the audio frames with the CM-Stick is shorter than the actual play time of the audio sample. In conclusions, the CM-Stick (due to its faster co-processor and USB interface) is able to provide more efficiency to StreamTo.

9.7 Related Work

In this section, we discuss related work.

9.7.1 Content Protection Approach

Several content protection approaches have been proposed. As some of them are proprietary, we are only able to scratch the surface.

Microsoft Secure Audio Path (SAP) [Microsoft, 2001] is a content protection approach specifically for audio. SAP tries to plug the "digital hole". When a user attempts to play an audio file, which is encrypted by an SAP-based method, the audio file is sent to a player where it is decrypted. Some audio noise is added to the decrypted audio signal and is removed only when the content output device (e.g. sound card) has been authenticated. Therefore, even if an attacker is able to intercept the decrypted audio signal, she will hear noise.

Digital Transmission Content Protection (DTCP) [DTCP, 1998] is proposed by Hitachi, Intel, Matsuhita, Sony and Toshiba. DTCP proposes a full-fledged content protection architecture for CE devices, which consists of compliant components. Several communication protocols for these components have been proposed to achieve key management, authentication, encrypted content transmission, content en/decryption etc., so that the content is protected throughout its cycle of usage.

DTCP is a hardware-based approach. Similar to SAP, DTCP attempts to patch the "digital hole" by protecting the transmission of the actual content between devices. For using the DTCP, a certain amount of royalty fees is required.

Both the SAP and DTCP rely on the authentication and trustworthiness of the device components. Therefore, a trusted computing platform (TCP) [Pearson et al., 2003] is a critical foundation to their approach. At this moment, to plug the "digital hole", TCP is deemed to be one of the potential solutions, which is discussed in the subsequent section.

9.7.2 Trusted Platform

A group of organizations, including Intel, IBM, and HP have formed the Trusted Computing Group (TCG). TCP tries to build a trusted and tamper-resistant system platform, e.g. a PC, a personal device assistant (PDA) etc., by adding tamper-resistant chips on the motherboard. These chips, the so-called Trusted Platform Module (TPM) and Core Root of Trust Module (CRTM) are used to store sensitive information of users, e.g. private keys, and the hash value of the current status of the platform.

Before any further processing on the platform is done, the current status of the platform is validated with the hash value stored on the chip. If a difference is detected, it might imply potentially unauthorized manipulation of the platform.

However, TCP technologies may induce severe inconvenience to users, e.g. the protected audio is only allowed to play on TCP-enabled PC but not other portable devices. It implies that the protected content is tightly coupled to the TCP platform because it is troublesome to move the secret key securely (once the key leaves the hardware, the physical security of the key cannot be guaranteed). Our method of using a hardware token provides a more user-friendly content usage (e.g. users can move and use their content "anyway and anywhere").

9.8 Conclusions and Future Work

We propose a streaming content protection approach, the so-called StreamTo, which combines the technology of the Internet streaming mechanism (SM), Pay-TV Conditional Access System (CAS) and a tamper-resistant hardware token.

StreamTo is able to bring tried and tested Pay-TV technology to the

personal computer (PC). In addition, StreamTo is able to protect while accessing the streaming content *offline*, which benefits the user (she can access the content without Internet connection), as well as the provider (she does not need to worry about overloaded servers when there is a large number of users). We analyze the security of StreamTo with respect to common security threats. We conclude that StreamTo is more secure than SM and CAS.

We implement StreamTo on two commercial tokens, namely the iButton and the CM-Stick, by using the CTR-mode of symmetric encryption. Thus, we show the applicability of StreamTo. We also evaluate the performance of the implementation to justify the practicality of StreamTo. The CM-Stick has a better performance than the iButton due to its faster cryptographic co-processor and USB interface.

Our future work is: (1) To integrate StreamTo with a standard streaming algorithm, such as the Real Time Streaming Protocol (RTSP). Thereby, we can use StreamTo for more efficient Internet streaming; and (2) To evaluate the performance of StreamTo on the network, by taking into account some Internet parameters, such as the bandwidth latency and delay, etc.

Part IV
EPILOGUE

This part provides the conclusions of the thesis and suggestions for future research.

CHAPTER 10
CONCLUSIONS AND FUTURE WORK

The thesis starts with three questions (see pp.1):

1. How can we control the way in which digital content is used?
2. How can we guarantee that users abide by our rules when using the digital content?
3. How can we allow the users to use the content anyway they like without violating our rules?

We investigate these questions in a number of experiments. The result of these experiments is a logic-based language, namely LicenseScript. We have also designed, implemented, and evaluated when necessary an architecture, which consists of cryptographic protocols, software components, etc. to support LicenseScript in enforcing the rights.

We also conclude several principles of rights control, as discussed in the following section.

10.1 Principles of Rights Control

Two fundamental principles have emerged, as a result of the experiments presented in the thesis:

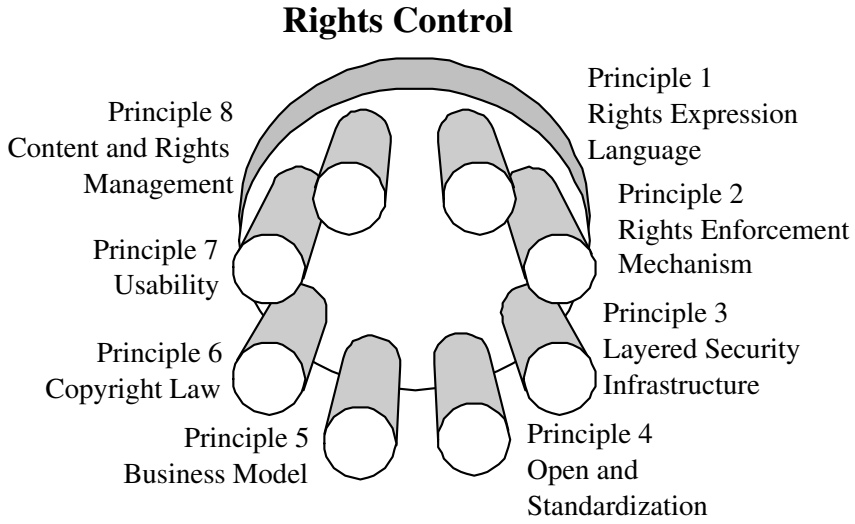

Figure 10.1: The principles of rights control systems (Rights Control Eight-Legged Stool).

Principle 1 *An expressive rights expression language is required.*

A rights expression language (REL) plays an important role in rights control. The language is used to construct a license, which specifies under which condition, what usage right can be exercised by a user on the content. A REL must be capable of providing fine-grained control on the content. Therefore, a REL must be expressive, flexible and extensible to express complex and novel usage scenarios.

Principle 2 *A tamper-resistant rights enforcement architecture is unavoidable.*

A tamper-resistant rights enforcement architecture is required to enforce the rights specified on the license correctly and securely. The components that constitute the rights enforcement architecture must be able to withstand attacks that a pure software solution on a standard personal computer cannot provide.

From our experiments, we conclude several additional general principles to support rights control, as shown in Figure 10.1:

Principle 3 *A layered security infrastructure of rights control is essential.*

Rushby [1981] suggests that "the security of a secure system should be achieved partly through the physical separation of its individual components and partly through the mediation of trusted functions performed within some of these components". A secure system should thus be extensible, flexible, distributed, simple and verifiable. To support more novel usage scenarios, which emerge frequently, the security infrastructure of a rights control system must be layered. For instance, the concept of authorized domain [van den Heuvel et al., 2002] is proposed to support home-network, where various consumer electronic (CE) devices are inter-networked for using the digital content.

Principle 4 *Open and standard solutions are required.*

There are many rights control solutions available, yet most of them are proprietary and incompatible with each other. This induces a heavy financial burden to providers and severe inconvenience to users. For instance, a DRM-enabled music piece of Windows Media Player cannot be played by a portable MP3 player. Therefore, an open and standard solution is required [Bremer and Buhse, 2003], which is able to (1) improve quality and reliability of rights control services; (2) increase compatibility and interoperability of rights control components and protocols; and (3) reduce the variety of data formats, metadata, algorithms and architectures, thus to attain economies of scale.

Principle 5 *A flexible and adaptable business model is required.*

A business model is "a unique blend of three streams that are critical to the business, which include the value stream for the business partners and the buyers, the revenue stream, and the logistical stream" [Buhse and Wetzel, 2003]. A business model also encapsulates the manner in which the content is distributed, such as centralized distribution (from the content provider to the users directly) or peer-to-peer distribution (among users themselves). The business model must be flexible to satisfy increasing demands of different

parties and adaptable to work seamlessly with the underlying rights control architecture.

Principle 6 *A universal and full-fledged copyright law is needed.*

Rights control requires the support from the copyright law, especially when there is a discovery of misuse of the copyrighted content. However, copyright law is diverse in different regions and immature in its support of digital content. The interplay between rights control and copyright law is intricate, since rights control can displace and override copyright law, while the law can constrain rights control [Samuelson, 2003]. Therefore, a universal and full-fledged copyright law is needed.

Principle 7 *Usability requires extensive care.*

Usability [Nielson, 1994] has become increasingly important in security [Yee, 2002]. A complex user-interface (UI) design or user's carelessness can always compromise the security [Whitten and Tygar, 1999]. For example, the use of passwords that can easily be guessed, or pin codes written on bank cards. Therefore, a user-friendly UI and sufficient education on using the system for users are necessary [Yee, 2002]. Furthermore, the rights control system should be easy to install, use and maintain so that the users will not feel uncomfortable.

Principle 8 *A flexible content and rights management is required.*

Content management [Rosenblatt and Dykstra, 2002] is critical to aid in building a rights control system. The lifecycle of content, i.e., creation, promotion, storage, distribution, and sale is particularly important if such rights control system is to generate revenue. Rights management, which includes rights creation, association (with the content), sharing, transfer, revocation, etc. is actually equally important. Usage control model [Sandhu and Park, 2003] is not yet able to support rights management because it is not capable of specifying administrative and delegation right.

As we have shown in the thesis, LicenseScript is a flexible and expressive rights expression language (Principle 1), and its underlying rights

enforcement architecture is able to achieve an extent level of tamper-resistance (Principle 2). Additionally, we have also shown that LicenseScript is capable of specify a wide variety of business models, thence LicenseScript is able to aid in Principle 5. LicenseScript is also able to model a useful copyright law, i.e., fair use, i.e., LicenseScript may be able to support Principle 7. We also have shown that LicenseScript is capable of supporting rights management, i.e., Principle 8.

The aforementioned principles are closely related to each other. For example, to fulfill Principle 7, Principle 4 is needed. Ultimately, the security of a rights control system can only be as good as the weakest link, which is formed by the users [Schneier, 2000]. There are different types of attackers [Abraham et al., 1991] who produce malicious tools to undermine the rights control systems. Users can easily obtain these tools via the Internet.

Currently, no one knows whether a balanced rights control system, which fulfills the aforementioned principles, that protects interests of users and the providers at large is ultimately feasible, from technological, business and legal perspective [Bechtold, 2003].

In short, the question of "can a 'good enough security' [Sandhu, 2003] for rights control ever exist" still requires extensive exploration.

10.2 Future Research

The thesis has explored two fundamental aspects of rights control (expressiveness and enforcement). In this chapter, we present research topics appropriate for further investigation.

Kernel-based protection. Principle 2 states the importance of tamper-resistance for rights enforcement components. We present a content protection approach at the application level in chapter 9. However, many believe that a kernel-based approach is the only proper solution to thorough and secure content protection, i.e., a solution with a secure operating system and a tamper-proof machine platform.

The trusted computing platform (TCP) [Pearson et al., 2003] has been proposed by Intel, HP, IBM and other giants. Meanwhile, Microsoft has

been developing the "next-generation secure computing base" (NGSCB, previously known as Palladium), which is composed of a trusted hardware architecture and a secure operating system. However, these solutions have not been used in practice due to several reasons: (1) Technologically, these solutions are still in their adolescent state and most likely will be incompatible to each other (which cannot satisfy Principle 4); and (2) Socially, users fear that these solutions will violate their privacy, and constrain their freedom of using their machines (which cannot achieve Principle 7). Therefore, a proven, technologically secure, and socially workable kernel-based content protection is required.

Authorized domain security. Most observers agree that home networking will only grow: Soon, a family's personal computer, television, set-top-box, stereo, DVD player, and other consumer electronic (CE) devices will be networked to form an authorized domain [van den Heuvel et al., 2002]. The content can be freely accessed, moved and copied amongst these CE devices. However, the same content cannot be accessed on domains other than the one for which it has been authorized. Therefore, the user is free to access and distribute the content within her authorized domain, while the content provider can protect her content from being illegally distributed.

Pestoni et al from IBM have provided rigorous implementation details for the authorized domain. They have proposed the eXtensible Content Protection (xCP) mechanism [Pestoni et al., 2004; Pestoni and Drews, 2003], which uses broadcast encryption [Fiat and Naor, 1994], to tackle the content and device management in an authorized domain. Several protocols have been designed in xCP to handle the security when devices join or leave the authorized domain. The main advantage of these protocols is that they exploit symmetric algorithm that can reduce resource consumption on the device.

However, other security-related issues of authorized domain, such as what happens when two authorized domains merge have not been addressed. This is closely related to Principles 3, 4, 5 and 7.

Interoperability and usability. To achieve Principles 4, 5 and 7, interoperability and usability of rights control must be considered carefully.

There is a growing agreement that interoperability and usability must become one of the principal attributes of a mature rights control infrastructure [Lyon, 2001].

Content providers sometimes attempt to get a large market share by producing proprietary products, which can be used to access only specific digital content. This constrains the users freedom significantly. For example, a Windows DRM-enabled MP3 cannot be played by other portable MP3 players. This has scared users from using these proprietary products, and the business involved might eventually suffer. Therefore, we require a solution that is able to achieve interoperability and compatibility amongst these products. This is another potential issue for future work.

As stated in Principle 7, security of the rights control should not compromise the usability, and vice versa [Whitten and Tygar, 1999]. It is not surprise that the installation, usage and maintenance of the rights control components can be arduous to users who do not possess sufficient technical knowledge. Therefore, it is challenging to find a balanced solution between security and usability for rights control. In short, by achieving interoperability and usability, users can use the digital content anywhere, anytime, and on any platform easily.

Value of protection to content or metadata. Our experiments show that the protection of license, content or metadata is highly resource-consuming in terms of processor power, memory, time, etc. This may inconvenience the users in using a rights control system [Chong et al., 2004; Cheng et al., 2004; Chong et al., 2002, 2003e]. For example, a user has to wait for 1 minute to create an audit log of playing a music.

Digital content can be of different value to different parties [Aichroth et al., 2004; Rosenblatt and Dykstra, 2002]: Some content value degrades after a short period of time, e.g. a stock price or breaking news; some content has value, which does not change for a longer period of time, e.g. film or music. Some content does not have any value as far as the user is concerned, but it is highly valuable to the content provider, e.g. audit logs. We need to determine the degree of protection required for the digital content according to its *current* value. Therefore, it is worthwhile to investigate the balance of the content value and the degree of protection, over a period of time.

This concludes our discussion on future research directions.

BIBLIOGRAPHY

M. Abadi. Logic in access control. In *18th Annual IEEE Symposium on Logic in Computer Science*, pages 228–232. IEEE Computer Society Press, June 2003.

D. G. Abraham, G. M. Dolan, and G. P. Double. Transaction security system. *IBM System Journal*, 30(2):206–229, 1991.

P. Aichroth, S. Puchta, and J. Hasselbach. Personalized previews: An alternative concept of virtual goods marketing. In *Int. Workshop for Technology, Economy, Social and Legal Aspects of Virtual Goods*, pages 10 pages (91–100), Ilmenau, Germany, May 2004. URL `http://virtualgoods.tu-ilmenau.de/2004/`.

R. Anderson and M. Kuhn. Low cost attacks on tamper-resistant devices. In B. Christianson, B. Crispo, T. Mark A. Lomas, and M. Roe, editors, *Security Protocols, 5th Internation Workshop Proceedings*, volume 1361 of *LNCS*, pages 125–136. Springer-Verlag, April 1997.

K. R. Apt. *From Logic Programming to Prolog*. Prentice Hall, Hertfordshire, United Kingdom, 1997.

K. R. Apt and D. Pedreschi. Reasoning about termination of pure Prolog programs. *Information and Computation*, 106(1):109–157, 1993.

M. J. Atallah and J. Li. Enhanced smart-card based license management. In *IEEE International Conference on E-Commerce*, pages 111–119. IEEE Computer Society, June 2003.

D. Aucsmith. Tamper resistance software: An implementation. In *Proceedings of First International Workshop on Information Hiding*, volume 1174 of *LNCS*, pages 317–333. Springer-Verlag, 1996.

J-P. Banâtre, P. Fradet, and D. L. Métayer. Gamma and the chemical reaction model: Fifteen years after. In C. Calude, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Workshop on Multiset Processing (WMP)*, volume 2235 of *LNCS*, pages 17–44. Springer-Verlag, Berlin, August 2001.

S. Bechtold. The present and future of digital rights management – musings on emerging legal problems. In *Digital Rights Management – Technological, Economic, Legal and Political Aspects*, volume 2770 of *LNCS*, pages 597–654. Spriner-Verlag, 2003.

M. Bellare and B. S. Yee. Forward integrity for secure audit logs. Technical report, UC at San Diego, Dept. of Computer Science and Engineering, November 1997. URL `http://citeseer.nj.nec.com/bellare97forward.pdf`.

E. Bertino, S. Castano, E. Ferrari, and M. Mesili. Controlled access and dissemination of XML documents. In *Proceedings 2nd ACM Workshop on Web Information and Data Management (WIDM'99)*, pages 22–27, 1999.

M. Bichler and A. Segev. A brokerage framework for internet commerce. *Distributed and Parallel Databases: Special Issue on E-Commerce*, 7(2):133–148, April 1999.

A. Bossi, N. Cocco, and M. Fabris. Proving termination of logic programs by exploiting term properties. In S. Abramsky and T.S.E. Maibaum, editors, *Theory and Practice of Software Development (TAPSOFT 91)*, volume 494 of *LNCS*, pages 153–180, Brighton, United Kingdom, April 1991, 1991. Springer-Verlag.

O. Bremer and W. Buhse. Standardization in DRM – trends and recommendations. In E. Becker, W. Buhse, D. Günnewig, and N. Rump, editors, *Digital Rights Management: Technological, Economic, Legal and Political Aspects*, volume 2770 of *LNCS*, pages 334–343. Springer-Verlag, November 2003.

S. Brin, J. Davis, and H. Garcia-Molina. Copy detection mechanisms for digital documents. In *Proceedings of the ACM SIGMOD Annual Conference San José*, pages 398–409, May 1995.

M. Buchheit and R. Kügler. Secure music content standard – content protection with codemeter. In *4th Open Workshop of Interactive Music Network Multimedia MUSICNETWORK*, page Paper 10, September 2004. URL `http://www.interactivemusicnetwork.org/events/Fourth_OpenWorkshop_2004/%musicnetwork-xxxc.pdf`.

W. Buhse and A. Wetzel. Creating a framework for business models for digital content – mobile music as case study. In E. Becker, W. Buhse, D. Günnewig,

and N. Rump, editors, *Digital Rights Management: Technological, Economic, Legal and Political Aspects*, volume 2770 of *LNCS*, pages 279–295. Springer-Verlag, November 2003.

L. J. Camp. DRM: doesn't really mean digital copyright management. In *Proceedings of the 9th ACM conference on Computer and Communications Security*, pages 78–87. ACM Press, 2002.

J. Cheng, C. N. Chong, J. Doumen, S. Etalle, P. H. Hartel, and S. Nikolaus. StreamTo: Streaming content using tamper-resistant tokens. Technical Report TR-CTIT-04-47, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, November 2004.

D. M. Chess. Security issues in mobile code systems. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *LNCS*, pages 1–14. Springer-Verlag Berlin, 1998.

C. N. Chong, R. Corin, S. Etalle, P. H. Hartel, W. Jonker, and Y. W. Law. LicenseScript: A novel digital rights language and its semantics. In K. Ng, C. Busch, and P. Nesi, editors, *3rd International Conference on Web Delivering of Music (WEDELMUSIC)*, pages 122–129, Los Alamitos, California, United States, September 2003a. IEEE Computer Society Press.

C. N. Chong, R. Corin, S. Etalle, P. H. Hartel, and Y. W. Law. LicenseScript: A novel digital rights language. In *Int. Workshop for Technology, Economy, Social and Legal Aspects of Virtual Goods*, page Paper 11, Ilmenau, Germany., May 2003b. URL `http://www.ub.utwente.nl/webdocs/ctit/1/000000ba.pdf`.

C. N. Chong, S. Etalle, and P. H. Hartel. Comparing logic-based and XML-based Rights Expression Languages. Technical Report TR-CTIT-03-30, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, July 2003c. URL `http://www.ub.utwente.nl/webdocs/ctit/1/000000cd.pdf`.

C. N. Chong, S. Etalle, and P. H. Hartel. Comparing Logic-based and XML-based Rights Expression Languages. In R. Meersman and Z. Tari, editors, *Proceedings of On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops*, volume 2889 of *LNCS*, pages 779–792, Berlin, Germany, November 2003d. Springer-Verlag.

C. N. Chong, S. Etalle, P. H. Hartel, R. Joosten, and G. Kleinhuis. Service brokerage with Prolog. In *Proceedings of 7th International Conference on Enterprise Information Systems (ICEIS 2005)*, page To appear. INSTICC Press, May 2005.

C. N. Chong, Z. Peng, and P. H. Hartel. Secure audit logging with tamper-resistant hardware. In D. Gritzalis, S. D. C. di Vimercati, P. Samarati, and S. K. Katsikas, editors, *18th IFIP International Information Security Conference (IFIPSEC)*, volume 250 of *IFIP Conference Proceedings*, pages 73–84. Kluwer Academic Publishers, May 2003e.

C. N. Chong, B. Ren, J. Doumen, S. Etalle, P. H. Hartel, and R. Corin. License protection with a tamper-resistant token. In C. H. Lim and M. Yung, editors, *5th Workshop on Information Security Applications (WISA 2004)*, volume 3325 of *LNCS*, pages 224–238. Springer-Verlag, August 2004.

C. N. Chong, R. van Buuren, P. H. Hartel, and G. Kleinhuis. Security attribute based digital rights management (SABDRM). In F. Boavida, E. Monteiro, and J. Orvalho, editors, *Joint Int. Workshop on Interactive Distributed Multimedia Systems/Protocols for Multimedia Systems (IDMS/PROMS)*, volume 2515 of *LNCS*, pages 339–352. Springer-Verlag, November 2002.

B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, page To appear, June 2003. URL `http://bitconjurer.org/BitTorrent/bittorrentecon.pdf`.

C. Collberg, C. Thomborson, and D. Low. Manufacturing cheap, resilient, and stealthy opaque constructs. In *25th Principles of programming languages (POPL)*, pages 184–196, San Diego, California, United States, January 1998. ACM Press, New York.

R. Corin, C. N. Chong, S. Etalle, and P. H. Hartel. How to pay in LicenseScript. Technical Report TR-CTIT-03-31, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, July 2003.

R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In M. V. Hermenegildo and G. Puebla, editors, *9th International Static Analysis Symposium (SAS)*, volume 2477 of *LLNCS*, pages 326–341. Springer-Verlag, September 2002.

E. Damiani, S. de C. di Vimercati, S. Paraboschi, and P. Samarati. Design and implementation of an access control processor for XML documents. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 33:59–75, June 2000a.

E. Damiani, S. de C. di Vimercati, S. Paraboschi, and P. Samarati. Securing XML documents. In *Advances in Database Technology - EDBT 2000, Proceedings 7th International Conference on Extending Database Technology*, volume 1777 of *Lecture Notes of Computer Science*, pages 121–135. Springer, March 2000b.

D. De Schreye and S. Decorte. Termination of logic programs: the never-ending story. *Journal of Logic Programming*, 19–20:199–260, 1994.

F. T. H. den Hartog, N. H. G. Baken, D. V. Keyson, J. J. B. Kwaaitaal, and W. A. M. Snijders. Tackling the complexity of Residential Gateway in an unbundling value chain. In *Proceedings of XVth International Symposium on Services and Local AccesS (ISSLS 2004)*, page To appear. IEE, March 2004.

J. DeTreville. Binder, a logic-based security language. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 105–113. IEEE Computer Society, March 2002.

R. M. Dijkman, L. F. Pires, and S. M. M. Joosten. Calculating with Concepts: a technique for the development of business process support. In A. Evans, R. France, A. Moreira, and B. Rumpe, editors, *Proceedings of the UML 2001 Workshop on Practical UML-Based Rigorous Development Methods*, volume 7 of *Lecture Notes in Informatics*, pages 87–98. GI-Edition, October 2001. ISBN 3-88579-335-0.

J. Dittman, A. Behr, M. Stabenau, P. Schmitt, J. Schwenk, and J. Ueberberg. Combining digital watermarks and collusion secure fingerprints for digital images. In *IEE Electronics and Communications, London*, pages 6/1–6/6, 2000.

DTCP. 5C digital transmission content protection – white paper. Technical report, Hitachi, Intel, Matsushita, Sony and Toshiba, July, 14 1998. URL `http://www.dtcp.com/`.

G. Durfee and M. Franklin. Distribution chain security. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 63–70. ACM Press, New York, 2000.

A. M. Eskicioglu and E. J. Delp. An overview of multimedia content protection in consumer electronics devices. *Signal Processing: Image Communication*, 16:681–699, 2001.

S. Etalle and M. Gabbrieli. Layered modes. *The Journal of Logic Programming*, 39(1–3):225–244, 1999.

S. Farrell and R. Housley. *An Internet Attribute Certificate Profile for Authorization, IETF Draft*. PKIX Working Group, January 2001. URL `http://search.ietf.org/internet-drafts/draft-ietf-pkix-ac509prof-09.txt`.

J. Feghhi and P. Williams. *Digital Certificates: Applied Internet Security*. Addison-Wesley, October 1998.

E. W. Felten. A skeptical view of DRM and fair use. *Communications of ACM*, 46(4):57–59, April 2003.

A. Fiat and M. Naor. Broadcast encryption. In *Advances in Cryptology (CRYPTO'03) Proceedings*, volume 773 of *LNCS*, pages 480–491, Santa Barbara, California, 1994. Springer-Verlag.

P. Fletcher, M. Waterhouse, and M. Clark. *Web Services Business Strategies and Architectures*. APress, July 2003.

J. Fridrich. Robust digital watermarking based on key-dependent basis functions. In *Proceedings of Second International Workshop on Information Hiding, USA*, pages 143–157, 1998.

B. Gelbord, H. Hut, G. Keinhuis, and E. Kwast. Access control based on attribute certificates. In *Proceedings of the IADIS International Conference WWW/Internet 2002 (ICWI 2002)*, pages 283–290. IADS, November 2002.

D. M. Goldschlag and D. W. Kravitz. Beyond cryptographic conditional access. In *USENIX Workshop on Smartcard Technology*, pages 87–91. USENIX Association, May 1999.

D. Gollmann. *Computer Security*. John Wiley & Sons, 1999.

J. Goshi and R. E. Ladner. Algorithms for dynamic multicast key distribution trees. In *Proceedings of of the twenty-second annual symposium on Principles of distributed computing*, pages 243–251. ACM Press, 2003.

T. C. Greene. MS digital rights management scheme cracked. *TheRegister.co.uk*, October 2001. URL `http://www.theregister.co.uk/content/4/22354.html`.

A. R. Greenwald and J. O. Kephart. Shopbots and pricebots. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 506–511. Morgan Kaufmann Publishers Inc., 1999.

L. Guibault. Copyright limitations and contracts: Are restrictive click-warp license valid? *Journal of Digital Property Law*, 2(1):144–183, November 2002.

L. C. Guillou. Smart cards and conditional access. In *Advances in Cryptology (EUROCRYPT 84)*, volume 209 of *LNCS*, pages 480–485. Springer-Verlag, 1984.

C. Gunter, S. Weeks, and A. Wright. Models and languages for digital rights. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, pages 4034–4038, Maui, Hawaii, United States, January 2001. IEEE Computer Society Press.

C. A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. MIT Press, 1992.

H. Guo. Digital rights management (DRM) using XrML. In *T-110.501 Seminar*

on Network Security 2001, page Poster paper 4, 2001. URL http://www.tml.hut.fi/Studies/T-110.501/2001/papers/.

R. H. Guttman and P. Maes. Agent-mediated integrative negotiation for retail electronic commerce. In *Selected Papers from the First International Workshop on Agent Mediated Electronic Trading Agent Mediated Electronic Commerce*, volume 1571 of *Lecture Notes in Computer Science*, pages 70–90. Springer-Verlag, 1998.

J. Haitsma and T. Kalker. A highly robust audio fingerprinting system. In *3rd International Conference on Music Information Retrieval (ISMIR)*, pages 107–115, 2002.

M. H. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1974.

F. Hartung and F. Ramme. Digital rights management and watermarking of multimedia content for m-commerce applications. *IEEE Communications Magazine*, 38(11):78–84, November 2000.

D. Henry. Who's got the key. In *Proceedings of the 27th annual SIGUCCS Conference on Mile high expectations*, pages 106–110, Denver, Colorado, United States, 1999. ACM Press.

B. Hillen, J. Kwaaitaal, A. van Neerbos, I. Passchier, and D. S. Rivero. Management requirements for residential gateways. Technical Report Deliverable D1.1 Project TSIT 1021, KPN Research and TU/e, The Netherlands, August 2002.

D. Holankar and M. Stamp. Secure streaming media and digital rights management. In *Proceedings of the 2004 Hawaii International Conference on Computer Science*, pages 85–96. ACM Press, January 2004.

G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

B. Horne, L. Matheson, C. Sheehan, and R. E. Tarjan. Dynamic self-checking techniques for improved tamper resistance. In *Workshop on Security and Privacy in Digital Rights Management 2001*, pages 141–159, February 2001a. URL http://www.star-lab.com/sander/spdrm/papers.html.

B. Horne, B. Pinkas, and T. Sander. Escrow services and incentives in peer-to-peer networks. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*, pages 85–94, October 2001b.

R. Iannella. Open digital rights management. In *World Wide Web Consortium (W3C) DRM Workshop*, page Position paper 23, January 2001. URL http://www.w3.org/2000/12/drm-ws/pp/.

P. C. Jain, S. Joshi, and V. Mitra. Conditional access in digital television. In *The 8th National Conference Communications (NCC) 2002*, page Technical Session paper 30, January 2002. URL http://www.ee.iitb.ac.in/uma/~ncc2002/proc/NCC-2002/.

R. Joosten, J-W. Knobbe, P. Lenoir, H. Schaafsma, and G. Kleinhuis. Specifications for the rge security architecture. Technical Report Deliverable D5.2 Project TSIT 1021, TNO Telecom and Philips Research, The Netherlands, August 2003.

P. Judge and M. Ammar. The benefits and challenges of providing content protection in peer-to-peer systems. In *Int. Workshop for Technology, Economy, Social and Legal Aspects of Virtual Goods*, page 12 pages, Ilmenau, Germany, May 2003. URL http://virtualgoods.tu-ilmenau.de/2003/.

T. Kalker, D. H. J. Epema, P. H. Hartel, R. L. Lagendijk, and M. van Steen. Music2Share - Copyright-Compliant music sharing in P2P systems (invited paper). *Proceedings of the IEEE Special Issue on Digital Rights Management*, 92(6):961–970, June 2004.

J. Kelsey and B. Schneier. Authenticating secure tokens using slow memory access (extended abstract). In *USENIX Workshop on Smart Card Technology*, pages 101–106. USENIX Press, 1999.

Kingpin. A practical introduction to the dallas semiconductor ibutton. Technical report, @Stake, Inc., 2002. URL http://www.atstake.com/research/reports/acrobat/practical_introduction_%to_ibutton.pdf.

D. W. Kravitz and D. M. Goldschlag. Conditional access concepts and principles. In *Proceedings of the 3rd International Conference on Financial Cryptography*, volume 1648 of *LNCS*, pages 158–172. Springer-Verlag, 1999.

C. Kruegel. *Network Security and Secure Applications: The Industrial Information Technology Handbook*. CRC Press, May 2004. ISBN 0-849-31985-4.

M. Kudo and S. Hada. XML document security based on provisional authorization. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 87–96, 2000.

S. H. Kwok and S. M. Lui. A license management model to support B2C and C2C music sharing. In *10th International World Wide Web Conference, Hong Kong*, pages 136–137, May 2001.

F. LaMonica. Streaming media. *Linux Journal*, 81es(6), January 2001.

L. Lamport. Password authentication with insecure communication. In *Communications of the ACM*, volume 24, pages 770–772. ACM Press, November 1981.

B. Lampson. Protection. *ACM Operating Systems Reviews*, 8(1):18–24, January 1974.

N. Li and M. V. Tripunitara. Security analysis in role-based access control. In *Proceedings of the 9th ACM symposium on Access control models and technologies*, pages 126–135. ACM Press, 2004. ISBN 1-58113-872-5.

D. Lie, C. Thekkath, M. Mitchell, and P. Lincoln. Architectural support for copy and tamper resistant. In *Architectural Support for Programming Languages and Operating Systems*, pages 168–177, 2000.

J. Linn. Attribute certification: An enabling technology for delegation and role-based controls in distributed environments. In *Proceedings of the 4th ACM Workshop on role-based access control on Role-based access control*, pages 121–130, Fairfax, Virgina, United States, 1999. ACM Press.

H. Lipmaa and P. Rogaway. Comments to NIST concerning AES-modes of operations: CTR-mode encryption. In *Symmetric Key Block Cipher Modes of Operation Workshop*, page Electronic Proceedings, October 2000. URL `http://www.tcs.hut.fi/~helger/papers/lrw00/`.

J. W. Lloyd. *Foundations of Logic Programming*. Symbolic Computation – Artificial Intelligence. Springer-Verlag, 1987. Second edition.

G. Lowe. Breaking and fixing the Needham-Schroeder Public-Key protocol using FDR. *Software Concepts and Tools*, 17:93–102, 1996.

G. Lyon. The Internet marketplace and digital rights management. *National Institute for Standards and Technology*, June 2001. URL `http://www.itl.nist.gov/div895/docs/GLyonDRMWhitepaper.pdf`.

B. M. Macq and J.-J Quisquater. Cryptology for digital tv broadcasting. *Proceedings of IEEE*, 83(6):944–957, 1995.

J. McCormac. *European Scrambling Systems*. Waterford University Press, 1996.

J. McLean. Security models. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. John Wiley & Sons, 1994. URL `http://citeseer.ist.psu.edu/mclean94security.html`.

A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*, chapter 12. CRC Press, 2001.

Microsoft. Understanding secure audio path. Technical report, Microsoft, 2001. URL `http://www.microsoft.com/windows/windowsmedia/drm/whitepapers.aspx`.

M. Mourad, J. Munson, T. Nadeem, G. Pacifici, and M. Pistoria. WebGuard: A system for web content protection. In *10th International World Wide Web*

Conference, Hong Kong, pages 142–143, May 2001.

D. Mulligan and A. Burstein. Implementing copyright limitations in rights expression languages. In J. Feigenbaum, editor, *Proceedings of 2002 ACM CCS-9 Workshop on Security and Privacy in Digital Rights Management*, volume 2696 of *LNCS*, pages 137–154. Springer-Verlag, November 2002.

D. K. Mulligan. Digital rights management and fair use by design. *Communications of ACM*, 46(4):31–33, April 2003.

J. Nielson. *Usability Engineering*. Academic Press, November 1994.

NIST. Secure hash standard. Technical Report FIPS PUB 180-1, US Department of Commerce/NIST, Washington D. C., United Stats, April 1995.

J. Park and R. Sandhu. The UCON$_{ABC}$ usage control model. *ACM Transactions on Information and System Security*, 7(1):128–174, February 2004.

D. Parrott. Requirements for a rights data dictionary and rights expression language. Technical Report version 1.0, Reuters Ltd., 85 Fleet St., London EC4P 4AJ, June 2001. In response to ISO/IEC JTC1/SC29/WG11 N4044: "Reissue of the Call for Requirements for a Rights Data Dictionary and a Rights Expression Language" – MPEG-21.

S. Pearson, B. Balacheff, L. Chen, D. Plaqui, and G. Proudler. *Trusted Computing Platforms – TCPA Technology in Context*. Prentice Hall PTR, Upper Saddle River, New Jersey 07458 United States, 2003.

F. Pestoni and C. Drews. eXtensbile Content Protection. In *Proceedings of the 11th ACM International Conference on Multimedia*, pages 458–459. ACM Press, 2003.

F. Pestoni, J. B. Lotspiech, and S. Nusser. xCP: Peer-to-peer content protection. *IEEE Signal Processing Magazine*, 21(2):71–81, March 2004.

R. Pucella and V. Weissman. A logic for reasoning about digital rights. In *IEEE Proceedings of the Computer Security Foundations Workshop*, pages 282–294, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society Press.

J. R. Rao, P. Rohatgi, H. Scherzer, and S. Tinguely. Partitioning attacks: Or how to rapidly clone some gsm cards. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 31–42. IEEE Computer Society, May 2002.

S. Ravi, A. Raghunathan, and S. Chakradhar. Tamper resistance mechanisms for secure embedded systems. In *IEEE International Conference on VLSI Design*, pages 605–614. IEEE Publisher, January 2004.

B. Rosenblatt and G. Dykstra. Integrating content management with digital rights management: Imperatives and opportunities for digital content lifecy-

cles. *GiantSteps Media Technology Strategies*, page 21 pages, May 2002. URL `http://www.giantstepsmts.com/drm-cm_white_paper.htm`.

B. Rosenblatt, B. Trippe, and S. Mooney. *Digital Rights Management: Business and Technology*. John Wiley & Sons, New York, United States, November 2002.

J. M. Rushby. Design and verification of secure systems. In *Proceedings of the eighth ACM symposium on Operating Systems Principles*, pages 12–21. ACM Press, 1981. ISBN 0-89791-062-1.

P. Samuelson. Digital rights management {and,or,vs.} the law. *Communications of ACM*, 46(4):41–45, April 2003.

R. Sandhu. The typed access matrix model. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 122–136, 1992.

R. Sandhu. Good-enough security: Towards a pragmatic business-driven discipline. *IEEE Internet Computing*, pages 66–68, January, February 2003.

R. Sandhu and J. Park. Towards usage control models: beyond traditional access control. In *7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, pages 57–64. ACM, June 2002.

R. Sandhu and J. Park. Usage control: A vision for next generation access control. In V. Gorodetsky, L. J. Popyack, and V. A. Skormin, editors, *Computer Network Security, 2nd International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security (MMM-ACNS'03)*, volume 2776 of *LNCS*, pages 17–31. Springer-Verlag, September 2003.

R. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, 32(9):40–48, September 1994.

B. Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption, Cambridge Security Workshop Proceedings*, pages 191–204. Springer-Verlag, December 1994.

B. Schneier. *Applied Cryptography, Second Edition*, chapter 15, pages 357–368. John Wiley & Sons, Inc., 1996.

B. Schneier. *Secrets and Lies*. John Willey & Sons Inc, 2000.

B. Schneier and J. Kelsey. Cryptographic support for secure logs on untrusted machines. In *The 7th USENIX Security Symposium Proceedings*, pages 53–62. USENIX Press, January 1998.

B. Schneier and J. Kelsey. Secure audit logs to support computer forensics. In *ACM Transactions on Information and System Security*, volume 2, pages 159–176. ACM Press, May 1999.

D. Sellars. An introduction to steganography. Technical report, University of Cape Town, Computer Science, May 1999. URL http://www.dsse.ecs.soton.ac.uk/techreports/99-4.html.

M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory. The british nationality act as a logic program. *Communications ACM*, 29(5):370–386, May 1986.

W. Shapiro and R. Vingralek. How to manage persistent state in DRM systems. In *Proceedings of the ACM Workshop in Security and Privacy in Digital Rights Management*, pages 176–191, November 2001. URL http://www.star-lab.com/sander/spdrm/papers.html.

C. Shi and B. Bhargava. A fast MPEG video encryption algorithm. In *Proceedings of the 6th ACM International Conference on Multimedia*, pages 81–88, Bristol, United Kingdom, 1998. ACM Press.

N. Shivakumar and H. Garcia-Molina. Building a scalable and accurate copy detection mechanism. In *Proceedings of the 1st ACM International Conference on Digital Libraries*, pages 160–168. ACM Press, March 1996.

O. Silbert, D. Bernstein, and D. van Wie. Digibox: A self-protecting container for information commerce. In *Proceedings of the First USENIX Workshop on Electronic Commerce*, pages 171–184, July 1995. URL http://citeseer.nj.nec.com/sibert95digibox.html.

L. Sterling and E. Shapiro. *The Art of Prolog (Second Edition)*, chapter 17, pages 319–357. The MIT Press, Cambridge, Massachusetts 02142, Uniter States, 1994.

S.A.F.A. van den Heuvel, W. Jonker, F.L.A.J. Kamperman, and P.J. Lenoir. Secure content management in authorised domains. In *Int. Broadcasting Convention (IBC)*, pages 467–474, Amsterdam, The Netherlands, September 2002. Broadcastpapers Pty Ltd, PO Box 259, Darlinghurst, NSW, 1300, AUSTRALIA.

S. Voloshynovskiy, S. Pereira, T. Pun, J. Eggers, and J. Su. Attacks on digital watermarks: Classification, estimation-based attacks and benchmarks. In *IEEE Communications Magazine (Special Issue on Digital Watermarking for Copyright Protection: a communications perspective)*, volume 39, pages 118–127. IEEE Press, 2001. M. Barni, F. Bartolini, I.J. Cox, J. Hernandez, F. Pérez-González, Guest Eds. Invited paper.

S. Wald. Secure media consumption in a Ubicomp World. In *Workshop on Security in Ubiquitous Computing (UBICOMP'02)*, page Paper 10, September 2002. URL http://www.teco.edu/~philip/ubicomp2002ws/.

A. Whitten and J. D. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Proceedings of 8th USENIX Security Symposium*, pages 169–184. USENIX, August 1999.

WIBU. *CodeMeter Developer's Guide*. WIBU-SYSTMES AG, Rueppurrer Str.53-54 76137 Karlsruhe, Germany, 1.0 edition, November 2003.

S-J. Yang and H-C. Chou. Adaptive QoS parameters approach to modelling Internet performance. *International Journal of Network Management*, 13:69–82, 2003.

Ka-Ping Yee. User interaction design for secure systems. In *Proceedings of 4th International Conference on Information and Communications Security*, pages 278–290. Springer-Verlag, 2002.

INDEX

Titles in the Telematica Instituut Fundamental Research Series

(sse also: `http://www.telin.nl/publicaties/frs.htm`)

001 G. Henri ter Hofte, *Working apart together: Foundations for component groupware*

002 Peter J.H. Hinssen, *What difference does it make? The use of groupware in small groups*

003 Daan D. Velthausz, *Cost-effective network-based multimedia information retrieval*

004 Lidwien A.M.L. van de Wijngaert, *Matching media: information need and new media choice*

005 Roger H.J. Demkes, *COMET: A comprehensive methodology for supporting telematics investment decisions*

006 Olaf Tettero, *Intrinsic information security: Embedding security issues in the design process of telematics systems*

007 Marike Hettinga, *Understanding evolutionary use of groupware*

008 Aart T. van Halteren, *Towards an adaptable QoS aware middleware for distributed objects*

009 Maarten Wegdam, *Dynamic reconfiguration and load distribution in component middleware*

010 Ingrid J. Mulder, *Understanding Designers, Designing for Understanding*

011 Robert Slagter, *Dynamic Groupware Services: Modular Design of Tailorable Groupware*

012 Nikolay Diakov, *Monitoring Distributed Object and Component Communication*

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-01

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-02

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-03

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-04

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-05

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-06

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-07

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-08

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-09

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty

of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Scheduler Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery proto-*

cols. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03