LicenseScript – A Language and Framework for Calculating Licenses on Information over Constrained Domains

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

September 27, 2002

Abstract

The project LicenseScript develops and demonstrates an integrated framework for analysis and design of secure information delivery systems. The contributions of this project are the demonstration of new analysis and design methods in at least two of the following concrete application areas: (1) Digital Rights Management (DRM), (2) wireless sensor networks (WSNs), (3) privacy protection.

In this paper, we introduce the motivation, concepts and theory of LicenseScript. More importantly we speculate and discuss the practical applicability of LicenseScript in the particular areas of DRM and WSNs. In fact, LicenseScript is borne out of our initially disjoint research efforts in DRM and WSNs. It is exciting that at the convergence point of the security requirements of DRM and WSNs, we found our solution: LicenseScript. In other words, LicenseScript is borne out of real needs, rather than an academic whim that seek a unified framework.

This paper is organized as follows. After the introduction in Section 1, we sketch the technical approach of LicenseScript in Section 2. Section 3 contains our discussion of the applicability of LicenseScript. Section 4 finishes off with our conclusion.

1 Introduction

Problems with establishing and enforcement of conditions of use on information are rife. For example the conditions of use might be internally inconsistent; they might be ambiguous, or simply incapable of expressing what an information provider wants. The conditions of use can also be deliberately misinterpreted or simply ignored, thus necessitating policing and necessarily the gathering of legal evidence. For conditions of use to be usable security tools, all these issues need to be addressed, which is precisely what the present project aspires to.

1.1 Introduction to Licensing

Most information, such as books, music, video, but also personal data and sensor readings, is intended for specific use. We believe that the conditions of use can always be governed by a license. Therefore, we propose to design a generic framework in which licenses can be modelled, and where new licenses can be created or calculated from existing licenses according to precisely defined rules. Within the framework, the conditions of use can either be checked when the information is used (online), or the conditions of use can be checked before or after use (off-line). A license governing some information typically describes two aspects: (1) what operations are valid on that information, and (2) under what circumstances operations are valid. Circumstances would include place, time, activity etc. We intend to capture both aspects (operations and circumstances) by the notion of *constraints*. Some constraints may be applied statically, thus reducing to types, whereas other constraints must be checked dynamically. Summarising: a license is data, an operation is a function on licenses and a domain is a constraint/type.

Operationally, a LicenseScript describes the roles of three types of actors: the information providers (e.g. content providers, or a person filling in a web form with personal data), the information consumers (e.g. consumers, or office workers) and the information monitors (e.g. intrusion detection, policing). Actors for each of the three different roles have limited trust in eachother, but we do not make a priori assumptions about who is trusted least (or most). However, typically the information consumers are trusted least and the information monitors would be trusted most.

Methodologically, a LicenseScript framework will consist of a number of components. The first component is the LicenseScript language, which will be used to describe at a high level of abstraction precisely what the intentions of the information providers are, what the potential actions of the information consumers are, and what measures can be taken by a monitoring system. The second component consists of a verification system (built on top of state-of-the-art verification technology) that will enable system designers to study the ramifications of their designs with regard to the policies, licenses, user actions and monitoring activity. In a typical verification step, potential user activity is verified against specifications of security objectives. The third component is a set of guidelines and examples showing how concrete designs can be developed, implemented and verified.

1.2 Objectives

The aim of the project is to contribute to a system level approach towards building secure information licensing systems. To achieve our aim we have identified the following concrete objectives:

1. To develop the LicenseScript language, tools and the theoretical underpinning that can be used to formalise the processes of licensing information in the broadest sense of the word.

2. To develop demonstrators for information providers using the language and tools, as well as off-the-shelf components for the realisation of the demonstrators.

3. To develop demonstrators for information monitors, again using the language and tools, and off-the-shelf components.

1.3 State of the Art

There are two technical approaches (discussed below) to protecting intellectual property in the digital world. These approaches are complementary and one of the ideas of this paper is to exploit that complementarity.

Secure Perimeter Schemes The first approach is to prevent the content from escaping the control of the Rights Holder. The most common technology is DRM, which packages the content with the rights pertaining to that content into some form of protecting digital envelope. The main problem with DRM is that at some stage the content will have to appear in clear text form so that the Consumer can hear, see and/or interact with the content. 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 coax into revealing content against the wishes of the Rights Holder. One should therefore expect that sooner or later the protection offered by DRM would be broken.

Traitor Tracing This is when the second class of technologies comes into the picture, i.e. the technologies that link the Rights Holder, the Distributor of the content, and/or the Consumer to the content itself. If such a link can be established, it will be possible to trace content that has ended up in the wrong hands. This is the area of traitor tracing [7]. The fact that such a link exists would hopefully act as a deterrent to copyright Infringers. Digital water marking and finger printing are sample technologies used for this linking purpose, but string searching might also be effective for some types of content.

LicenseScript tries to embrace the two subject areas, by providing a framework in which designers are able to reason about the use the various technologies, in isolation and in combination. We are thus not primarily interested in developing new DRM techniques, or new cryptographic techniques; we are interested in abstracting from the techniques, focusing on the essential aspects.

Gunter et al. [8] characterise a license by a trace generated from a simple language. Checking that an action conforms to a particular license means checking whether the action occurs in the trace. Pucella and Weissman [12] cast Gunter et al's work in a modal logic framework. Further survey of other related work can be found in Appendix A.

2 Our Approach

We begin the description of our approach with an example of the power we expect of LicenseScript. The notation we use is *multiset rewriting* (a multi set is a set in which elements are not necessarily unique). Basically, multiset rewriting is the notation of the coordination language Gamma [1]. In [3] we have already employed an extension of multiset rewriting for the specification and the verification of security protocols.

2.1 Multiset Rewriting

The core of a license consists of a multiset `M` and of a set of rules `R`. The multiset can be regarded as the memory where data is stored, and the rules determine the real meaning of the license. To each `EVENT` (e.g. `play`, `pay`, `license split`) a rule is associated. The rule determines whether the event can take place and what the state of the multiset should be after the event has taken place. For the sake of simplicity, we assume here that events are atomic.

The rules we use have the form

```
MUTLTISET1 -> MULTISET2 | CONDITION
```

To illustrate (with heavily simplified examples) how powerful these rules are, we have to establish a minimal notation for the multiset. Assume that `lic(mus,n)` is a term indicating that the song `m` can be played another `n` times. A very simple rule associated to the event `play` would be:

```
play:  lic(mus,n) -> lic(mus,n-1) | n > 0
```

Let us now add a new global constant: `this` is the constant that refers to the environment containing the license (e.g. the appliance where the music is stored). In particular `this.date` is the current date, `this.signature` is the signature of the device. We can then extend the `play` rule with a deadline:

```
play:    lic(mus,n), expires(time)
      -> lic(mus,n-1), expires(time)
       | n > 0, time < this.date
```

We can also express the fact that a license is valid only for the appliance where it was initiated on. This is done by the following two rules:

```
init: device(x) -> device(this.signature) | x = any

play:     lic(mus,n),  device(x)
       -> lic(mus,n-1),  device(x)
       |  x = this.signature,
          n > 0
```

The first rule states that initially we accept any device, but thereafter the multiset item `device(x)` is replaced by the specific multiset item `device(this.signature)`. The second rule states that both the item `lic(mus,n)` and the item `device(x)` must be present in the multiset, and that the given constraint condition must be satisfied.

The following license allows playing a song only once in 24 hours (assuming that `this.date` is expressed in seconds):

```
play:     lic(mus,n),  last_play(t)
       -> lic(mus,n-1), last_play(this.date)
        | this.date > t + 24*3600,
          n > 0
```

A license can be split, so that a sub-license can eventually be moved to another authorized domain. The basic mechanism is the following:

```
split:     lic(mus,n), expires(time)
        -> lic(mus,n1), expires(time),
           lic(mus,n2), expires(time)
         | n1 > 0, n2 > 0, n1 + n2 = n
```

The idea is that the multiset of a license can be split any time, to form two or more sub-licenses.

The following two rules allow the owner of a (high quality) license to give away - once a day - a license for listening once the same song, but at a lower quality level (for instance, for publicity reasons):

```
split:     lic(mus,n), expires(time),
           last_splitting(time)
        -> lic(mus,n), expires(time),
           lic(lower_quality(mus),1),
           last_splitting(this.date)
         | n > 0,
           time < this.date - 24*3600
```

In the same way, one can define payment events. In the Gamma notation it is easy to model the payment methods `upfront`, `flatrate` and `peruse` employed by Gunter et al. [8]. In fact, it should be straightforward to demonstrate the paradigm presented here is strictly more expressive than the language DigitalRights of Gunter et al.

2.1.1 Authorized Domains

Authorised domain (AD), as described by van den Heuvel et al. in [16] can be implemented in a straightforward way within this framework. For instance, to represent a license that is bound to a domain, one could use the token `lic(mus,domain)`. The registration of a device within a given domain could simply take place by adding the token `belongs_to(domain_signature,expiration_time)` to the multiset. This addition must be carried out by the AD Device Manager (ADDM), as defined in [16]. For extra security, `domain_signature` can be a cryptographic key that binds `domain` to the signature of the device `this.signature`. A rule that allows playing a license only if the device belongs to the correct domain could be the following:

```
play:   lic(mus,domain),
        belongs_to(domain_signature,expiration_time)
     -> lic(mus,domain),
        belongs_to(domain_signature,expiration_time)
      | expiration_time < this.date,
        check_signature(domain,
                        domain_signature,
                        this.signature)
```

By default, a device can belong to multiple authorised domains. This can be implemented by allowing in the multiset multiple tokens of the form `belongs_to(domain_signature,expiration_time)`.

2.1.2 Correctness of Transformation

Of course, it is of foremost importance that the system is sound, i.e. that the various transformation operations cannot be combined in such a way that one could for instance listen to a song without having an appropriate license for it.

When looked at from the foundational viewpoint, the examples above suggest a framework for the transformation of a concurrent paradigm (in particular, Gamma). By providing a semantics to the paradigm, for instance in terms of the possible traces of actions (play, upgrade, etc) that can take place, we can address the problem of the soundness of the system by reducing it to the problem of the correctness of the transformation system with respect to the chosen semantics, which is a problem that readily leverages the results of Etalle et al. [5, 4, 6].

2.2 Verification

In the framework depicted above, there are two aspects that require the development and use of verification tools. In the first place, the Information Monitors should be provided with tools that enable them to verify the integrity of the licenses and that allow them to trace back the fraud sources. Secondly, experience shows that when one employs cryptographic means (which in this context will be absolutely necessary for the communication between the different devices and for checking the integrity of the system) subtle `mistakes` can allow ingenious attacks (e.g. 'man-in-the-middle'), that for instance can allow a malicious intruder to gather information he is not allowed to have.

We call the first aspect 'monitoring' and the second 'model checking'. We now outline them separately.

2.2.1 Monitoring

Basically, Information Monitors have to be able to check the INTEGRITY of the licenses. There are two sides of this problem that we will call respectively LOCAL and GLOBAL integrity. By LOCAL INTEGRITY we indicate the fact that one should be able to verify whether a license (in isolation) is a legal one, that is, whether it was issued by a legitimate entity and whether it has been tampered with. As soon as we allow algebraic operations on licenses, this task becomes non-trivial, as a license can change in time. Basically, we need a system that uses digital signatures and that records at least part of the 'history' of a license, so that its lawful origin can be verified. Here, it is of crucial importance that the system is simple and scalable: the simpler licenses (e.g., those for music to be played on Consumer Electronics devices) have to be of manageable size.

On the other hand, by GLOBAL INTEGRITY we indicate the fact that the Information Monitors should be able to verify whether a GROUP of licenses is legal or not. Consider the case in which a license L has been illegally duplicated (for instance, by tampering with the medium that carries L). Assume also that L is subsequently split in two sublicenses L1 and L2. After the splitting, neither the license L nor its copy should be used any longer. If by monitoring we find that both L (or its copy) and L1 are being used then the monitor should be able (a) to verify that an illegal situation has indeed been detected and possibly (b) to identify what is the source of the unlawful situation, that is, find out at which point of the evolution of the license the fraud was carried out.

2.2.2 Model Checking

Cryptographic protocols are the essential means for the exchange of confidential information and for authentication, and will necessarily form a structural part in the mechanism for license deployment.

Therefore, the correctness of the employed protocols will be crucial for guaranteeing that no hostile intruder will be able to get hold of secret information (e.g. a private key) or to force unjust authentication. Unfortunately, the design of cryptographic protocols is a well-known error-prone process. A great deal of published protocols has later been shown to contain error prejudicing their safety. In various cases such flaws were found years after the proposal of a protocol, demonstrating how subtle such errors can be. To cope with this problem, in recent years various protocol verifiers have been devised and deployed. In [2] we have developed a state-of-the-art system for the verification of two-party security protocols (at the moment the system of [2] is most likely by far the fastest of its kind of verifiers).

As we said, for the delivery of licenses we will have to develop new multi-party cryptographic protocols. To be sure of their correct working, these protocols will have to be verified

formally, and for this we need to extend in a non-trivial way the system of [2]. In particular, we have to cover the following two aspects. First, non-repudiation, i.e. the ability to demonstrate that a certain actor has committed to a certain action (e.g. to a payment). To capture non-repudiation we have to extend the tool we have with a game-based semantics, as shown in [9]. Secondly, we have to consider multi-part communication. In a context in which licenses can be split and recombined together, we cannot think of protocols as being carried out just by two honest principals and an intruder: the increased number of principals can allow for more complex attacks, in which different principals collude to subvert the orderly procedure of the protocol.

Summarising, the basic concepts of LicenseScript are clearly relevant and permit powerful specifications to be made. The theoretical underpinning in the form of linear logic, game semantics and cryptography is also well established. The major challenges lay in the integration of theses theories into a well-founded and practical tool.

3 Discussion

In this section, we look at the applicability of LicenseScript in DRM and WSNs. However due to space limitation, we can only afford to paint a rough picture of what intend to achieve.

3.1 LicenseScript and DRM

LicenseScript is closely related to DRM because the license (carrying rights and rules of using the content) is a main part of a DRM system. Up to this moment, a DRM development process is no different from any other system development process: establishing the requirements, planning and designing the system architecture, implementing and testing the prototypes of the system, and finally introducing the system to the market (and updating the system). Testing normally is a painful and costly task. By using LicenseScript, which is able to provide rich semantics and affordance for verification, in DRM system development, the pain can be alleviated.

There is currently a wide variety of business models in DRM [13], such as subscription-based DRM, music and movie preview, free trial of usage etc. LicenseScript can facilitate the development of DRM systems that support these different business models. In other words, LicenseScript is a tool for developing reliable *and* flexible DRM systems.

Several digital rights languages have been proposed lately, such as XrML (`http://www.xrml.org`), ODRL (`http://odrl.net`) and MPEG-7 (`http://mpeg.telecomitalialab.com`). They provide a rich syntax and structure that allows Rights Holders to specify fine-grained control over contents. The aforementioned languages are able to express different kinds of rights and a myriad of terms and conditions, but they do not have precise formal semantics – their interpretation relies solely on human intuition of the syntactic features. Additionally, digital licenses described using these rights languages tend to become syntactically complicated and complex when more entities, rights, and rules governing the usage of the contents are involved. LicenseScript is meant to be a simple and elegant language, that does not impose such disadvantages.

3.2 LicenseScript and WSNs

In the EYES project (`http://eyes.eu.org`), we are developing self-organizing and collaborative energy-efficient sensor networks. As sensors operate unattended and the communication transmissions occur in an open medium, the system is susceptible to physical attacks and network attacks. In [10], we reason that although a fundamental mechanism, cryptography is not enough for combating unscrupulous nodes (i.e. malicious nodes that contain legitimate cryptographic key material) and the malevolent army of colluding nodes. As a remedy, Marti et al. [11] propose monitoring the transmission of neighbouring nodes, on the network routing level. [17] propose running intrusion detection agents on every node. We like to generalize the above ideas into a unified framework of intrusion detection. For this purpose, it is necessary to compile a list of rules against which to check for anomaly. For example, from a data-centric point of view, a node should be suspected when it: (1) suddenly requests information that has never been requested before by anybody, or (2) suddenly requests information that itself has never requested before, or (3) requests information in an untypical order, or (4) requests information that it itself is supposed to supply, or (5) requests information that does not exist, or (6) requests information without proper clearance, or (7) requests information exceptionally frequently, and so on. The list would not seem to end, because what is anomalous depends on the definition of what is normal, which is different from application to application – the list is application-specific.

For this application-specificity, WSNs need some sort of mechanism for specifying the rights, restrictions, and level of clearance of each node, for each application. In other words, each node needs a "license" to execute certain functions. To enforce these licenses, a Supervisor is needed to make sure each node complies by the rules set out in their license, and that the violators be punished. This is where LicenseScript comes in. If policy is an analogy of "access control" [15, 14], then license is an analogy of "capabilities". Examples:

- A *resource usage license* dictates the amount and types of resources that are eligible for the license holder.
- A *private data license* dictates the scope of that private data that are entitled to the license holder.
- A *pattern generation license* dictates the rights of the holder to be immuned from detection, for generating particular intrusion patterns.

These different types of license can in fact be combined into one generic license, for simplicity and efficiency. License should be able to revoked. In other words, like real-world licenses, they are time-limited. All of these requirements will affect the design of LicenseScript.

4 Conclusion and Future Work

As a conclusion, we have outlined the rationale, concepts and philosophy behind LicenseScript. At the same time speculating the potential applications of LicenseScript, we take into consideration the particular requirements of DRM and WSNs. However between over-ambitiousness and simplicity, we will strive to make LicenseScript as simple, practical and useful as possible.

Our LicenseScript licenses can be likened to proofs that the executing code complies with the safety policy of the system, similar to the safety proofs in the context of Proof-Carrying Code (PCC) (`http://www.cs.princeton.edu/sip/projects/pcc/`). Therefore in stock is our plan to investigate further applications of LicenseScript in distributed programming languages.

References

[1] J.-P. Banâtre and D. Le Métayer. Programming by multiset transformation. *Communications of the ACM*, 36(1):98, Jan 1993.

[2] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In M. Hermenegildo and G. Puebla, editors, *Int. Static Analysis Symp. (SAS)*, page to appear, Madrid, Spain, Sep 2002. Springer-Verlag, Berlin.

[3] G. Delzanno and S. Etalle. Proof theory, transformations, and logic programming for debugging security protocols. In A. Pettorossi, editor, *Proc. Eleventh International Workshop on Logic Program Synthesis and Transformation – LOPSTR 2001*, LNCS, pages 76–91. Springer-Verlag, 2002.

[4] S. Etalle and M. Gabbrielli. Partial evaluation of concurrent constraint languages. *ACM Computing Surveys*, 30(3):electronic edition, Sep 1998.

[5] S. Etalle, M. Gabbrielli, and E. Marchiori. A transformation system for CLP with dynamic scheduling and CCP. In *Partial evaluation and semantics-based program manipulation (PEPM)*, pages 137–150, Amsterdam, The Netherlands, Jun 1997. ACM Press, New York.

[6] S. Etalle, M. Gabbrielli, and M.C. Meo. Transformations of ccp programs. *ACM Transactions on Programming Languages and Systems*, 23(3):304–395, 2001. To appear. Available also at CoRR http://arXiv.org/.

[7] A. Fiat and T. Tassa. Dynamic traitor tracing. *Journal of Cryptology*, 14(3):211–223, 2001.

[8] C.A. Gunter, S. Weeks, and A. Wright. Models and languages for digital rights. In *34th Annual Hawaii Int. Conf. on System Sciences (HICSS)*, page 9076, Maui, Hawaii, Jan 2001. IEEE Computer Society Press, Los Alamitos, California.

[9] S. Kremer and J-F. Raskin. A game-based verification of non-repudiation and fair exchange protocols. In K. G. Larsen and M. Nielsen, editors, *Proc. Concur'01, 12th International Conference of Concurrency Theory*, number 2154 in LNCS, pages 551–565. Springer-Verlag, 2001.

[10] Y.W. Law, S. Dulman, S. Etalle, and P. Havinga. Assessing Security-Critical Energy-Efficient Sensor Networks. Technical Report TR-CTIT-02-18, Department of Computer Science, University of Twente, Jul 2002.

[11] S. Marti, T.J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 255–265. ACM Press, 2000.

[12] R. Pucella and V. Weissman. A logic for reasoning about digital rights. In *15th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, Canada, Jun 2002. IEEE Computer Society Press, Los Alamitos, California.

[13] B. Rosenblatt, B. Trippe, and S. Mooney. *Digital Rights Management: Business and Technology*. John Wiley & Sons, 1st edition, Nov 2001.

[14] F. Stajano. The Resurrecting Duckling - what next? In *Proceedings of The 8th International Workshop on Security Protocols*, volume 2133 of *LNCS*, pages 204–214. Springer-Verlag, 2000.

[15] F. Stajano and R. Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In B. Christianson, B. Crispo, and M. Roe, editors, *Proceedings of The 7th International Workshop on Security Protocols*, LNCS, pages 172–182. Springer-Verlag, 1999.

[16] 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 *The World's Electronic Media Event IBC 2002, Sept. 13-17, Amsterdam RAI, The Netherlands*, Sep 2002. To appear.

[17] Y. Zhang and W. Lee. Intrusion detection in wireless ad-hoc networks. In *Proc. 6th Annual ACM/IEEE International Conference on Mobile Computing (MOBICOM'00)*, pages 275–283, 2000.