SUPPORT CONSUMERS’ RIGHTS IN DRM: A SECURE AND FAIR SOLUTION TO DIGITAL LICENSE RESELLING OVER THE INTERNET

A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy in the Faculty of Engineering and Physical Sciences

2012

By
Tarek Gaber
School of Computer Science
Contents

Abstract 17
Declaration 19
Copyright 21
Dedication 23
Acknowledgement 25
Abbreviations 27
Definitions 29
Notaitons 31

1 Introduction 33
  1.1 Introduction to Digital Rights Management (DRM) . . . . . . . . 33
  1.2 Introduction to Concurrent Signature Scheme . . . . . . . . . . . 34
  1.3 DRM and Consumers’ Rights . . . . . . . . . . . . . . . . . . . . 36
  1.4 Research Motivation and Challenges . . . . . . . . . . . . . . . . 38
  1.5 Research Aim and Objectives . . . . . . . . . . . . . . . . . . . . 40
  1.6 Research Methodology . . . . . . . . . . . . . . . . . . . . . . . 41
  1.7 Novel Contributions and Publications . . . . . . . . . . . . . . . 42
  1.8 Thesis Structure . . . . . . . . . . . . . . . . . . . . . . . . . . 46

2 Digital Rights Management Overview 47
  2.1 Chapter Introduction . . . . . . . . . . . . . . . . . . . . . . . . 47
  2.2 What is DRM . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
  2.3 DRM History . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.1</td>
<td>First Generation DRM Systems</td>
<td>49</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Second Generation DRM Systems</td>
<td>49</td>
</tr>
<tr>
<td>2.4</td>
<td>DRM Fundamental Principle</td>
<td>50</td>
</tr>
<tr>
<td>2.5</td>
<td>DRM System Entities</td>
<td>52</td>
</tr>
<tr>
<td>2.6</td>
<td>DRM System Components</td>
<td>53</td>
</tr>
<tr>
<td>2.7</td>
<td>How a DRM System Works</td>
<td>55</td>
</tr>
<tr>
<td>2.8</td>
<td>Existing DRM Systems</td>
<td>56</td>
</tr>
<tr>
<td>2.8.1</td>
<td>Windows Media DRM</td>
<td>56</td>
</tr>
<tr>
<td>2.8.2</td>
<td>FairPlay DRM</td>
<td>58</td>
</tr>
<tr>
<td>2.8.3</td>
<td>Open Mobile Alliance (OMA) DRM</td>
<td>61</td>
</tr>
<tr>
<td>2.9</td>
<td>Open Issues in Current DRM Systems</td>
<td>65</td>
</tr>
<tr>
<td>2.9.1</td>
<td>Consumer Privacy</td>
<td>65</td>
</tr>
<tr>
<td>2.9.2</td>
<td>Interoperability</td>
<td>66</td>
</tr>
<tr>
<td>2.9.3</td>
<td>First-sale: License Reselling</td>
<td>67</td>
</tr>
<tr>
<td>2.10</td>
<td>Chapter Summary</td>
<td>68</td>
</tr>
<tr>
<td>3</td>
<td>A literature Survey</td>
<td>71</td>
</tr>
<tr>
<td>3.1</td>
<td>Chapter Introduction</td>
<td>71</td>
</tr>
<tr>
<td>3.2</td>
<td>Current License Selling Solutions</td>
<td>71</td>
</tr>
<tr>
<td>3.2.1</td>
<td>License Selling Process in WM-DRM</td>
<td>72</td>
</tr>
<tr>
<td>3.2.2</td>
<td>License Selling Process in FairPlay</td>
<td>73</td>
</tr>
<tr>
<td>3.2.3</td>
<td>OMA DRM License Selling</td>
<td>74</td>
</tr>
<tr>
<td>3.3</td>
<td>Current License Reselling Solutions</td>
<td>77</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Full-trusted Hardware based Solutions</td>
<td>79</td>
</tr>
<tr>
<td>3.3.1.1</td>
<td>Kwok’s DRM System</td>
<td>79</td>
</tr>
<tr>
<td>3.3.1.2</td>
<td>Sun’s System</td>
<td>80</td>
</tr>
<tr>
<td>3.3.1.3</td>
<td>NPGCT DRM System</td>
<td>82</td>
</tr>
<tr>
<td>3.3.1.4</td>
<td>Nuovo DRM System</td>
<td>87</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Partial-trusted Hardware Based Solution: Conrado’s System</td>
<td>89</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Non-trusted Hardware based Solution: Laila’s System</td>
<td>91</td>
</tr>
<tr>
<td>3.4</td>
<td>What is Missing?</td>
<td>94</td>
</tr>
<tr>
<td>3.5</td>
<td>The Best Way Forward</td>
<td>96</td>
</tr>
<tr>
<td>3.6</td>
<td>Chapter Summary</td>
<td>97</td>
</tr>
<tr>
<td>4</td>
<td>A Reselling Deal Signing (RDS) Protocol</td>
<td>99</td>
</tr>
<tr>
<td>4.1</td>
<td>Chapter Introduction</td>
<td>99</td>
</tr>
</tbody>
</table>
## 4.8.4.1 RDS Protocol Design

4.8.4.2 RDS Protocol Implementation

4.8.4.3 Hardware and Software Architecture

4.8.4.4 RDS Protocol Evaluation

4.8.4.4.1 Performance Evaluation

4.8.4.4.2 Test against Security Attacks

### 4.9 Chapter Summary

## 5 Reselling Deal Method

### 5.1 Building Blocks

5.1.1 Market Power

5.1.2 Existing License Distribution Infrastructure

5.1.2.1 License Issuer (LI)

5.1.2.2 DRM Client

5.1.3 License Revocation List (LRL)

5.1.3.1 LRL Types

5.1.3.2 Delivering LRL to Reseller

5.1.3.2.1 Pull Mode

5.1.3.2.2 Push Mode

5.1.3.3 The Need for Imposing a Reselling Deal Validity Deadline

5.1.4 Reselling Permission (RP)

### 5.2 Preliminaries

5.2.1 Notations

5.2.2 Design Assumptions

5.2.3 Design Requirements

### 5.3 Fair and Secure License Reselling Protocol (FSLRP) Suite: An Overview

### 5.4 The FSLRP Protocol Suite in Detail

5.4.1 2-Messages RD Signing (2M-RDS) Protocol

5.4.1.1 2M-RDS Protocol Overview

5.4.1.2 2M-RDS protocol Analysis

5.4.2 RD Activation (RDA) Protocol

5.4.2.1 RDA Protocol Overview

5.4.2.2 RDA Protocol Analysis

5.4.3 RD Completion (RDC) Protocol

6
5.4.3.1 RDC Protocol Overview ........................................ 190
5.4.3.2 RDC Protocol Analysis ........................................ 191
5.5 Threat and Attack Analysis ......................................... 192
  5.5.1 Threats Analysis .................................................. 192
    5.5.1.1 Double Use of a License .................................. 193
    5.5.1.2 Installing an Out-of-date LRL-update .................... 194
  5.5.2 Attack Analysis .................................................. 194
    5.5.2.1 Collusion Attack 1 ......................................... 194
    5.5.2.2 Collusion Attack 2 ......................................... 195
5.6 FSLRP suite Evaluation ............................................. 196
  5.6.1 FSLRP Suite Computational Cost ............................... 196
  5.6.2 Comparison with Related Work ................................ 197
5.7 Chapter Summary ...................................................... 201

6 Two Methods Supporting Multi-Reselling ............................. 203
  6.1 Chapter Introduction ............................................... 203
  6.2 Additional Design Building Block ................................. 204
  6.3 Preliminaries ....................................................... 204
    6.3.1 Definitions ................................................... 204
    6.3.2 Design Assumptions .......................................... 206
    6.3.3 Design Requirements ......................................... 206
  6.4 Method One: Repeated RP based Multi-reselling ................. 207
    6.4.1 RRP-MR Method Overview ..................................... 208
    6.4.2 RRP-MR Method in Detail ..................................... 208
    6.4.3 Verifications used in the RRP-MR Method .................... 214
      6.4.3.1 Buyer’s Verifications .................................. 214
      6.4.3.2 LI’s verifications ....................................... 217
    6.4.4 RRP-MR Method Analysis ..................................... 219
      6.4.4.1 Analysis against Requirements ......................... 219
      6.4.4.2 Analysis against Potential Attacks .................... 220
      6.4.4.3 RRP-MR Method Weaknesses ............................... 222
  6.5 Method Two: Hash Chain based Multi-reselling ................. 222
    6.5.1 HC-MR Method Overview ...................................... 225
    6.5.2 Multiple Reselling Permission (MRP) ........................ 226
    6.5.3 Read-only Public Directory (RPD) for Multi-Reselling Check-
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5.4 HC-MR Method in Detail</td>
<td>229</td>
</tr>
<tr>
<td>6.5.5 Verifications Used in the HC-MR Method</td>
<td>235</td>
</tr>
<tr>
<td>6.5.5.1 Buyer’s Verifications</td>
<td>236</td>
</tr>
<tr>
<td>6.5.5.2 LI’s verifications</td>
<td>242</td>
</tr>
<tr>
<td>6.5.6 The HC-MR Method Analysis</td>
<td>243</td>
</tr>
<tr>
<td>6.5.6.1 Analysis against Requirements</td>
<td>243</td>
</tr>
<tr>
<td>6.5.6.2 Security Analysis against Potential Attacks</td>
<td>245</td>
</tr>
<tr>
<td>6.6 Evaluation of the Two Multi-reselling Methods</td>
<td>248</td>
</tr>
<tr>
<td>6.6.1 Computational Costs of the Two Methods</td>
<td>248</td>
</tr>
<tr>
<td>6.6.2 Comparison with Related Work</td>
<td>250</td>
</tr>
<tr>
<td>6.7 Chapter Summary</td>
<td>253</td>
</tr>
<tr>
<td>7 Conclusion and Future Work</td>
<td>255</td>
</tr>
<tr>
<td>7.1 Thesis Contributions</td>
<td>256</td>
</tr>
<tr>
<td>7.2 Directions for Future Work</td>
<td>260</td>
</tr>
<tr>
<td>7.3 Deployment Requirements</td>
<td>262</td>
</tr>
<tr>
<td>Bibliography</td>
<td>264</td>
</tr>
<tr>
<td>A All Protocols in Detail</td>
<td>281</td>
</tr>
<tr>
<td>A.1 RDS Protocol in Detail</td>
<td>281</td>
</tr>
<tr>
<td>A.2 2M-RDS Protocol in Detail</td>
<td>284</td>
</tr>
<tr>
<td>A.3 RDA Protocol in Detail</td>
<td>288</td>
</tr>
<tr>
<td>A.4 RDC Protocol in Detail</td>
<td>294</td>
</tr>
<tr>
<td>B Building Blocks Used in DRM</td>
<td>299</td>
</tr>
<tr>
<td>B.1 Chapter Introduction</td>
<td>299</td>
</tr>
<tr>
<td>B.2 Symmetric Cryptography</td>
<td>300</td>
</tr>
<tr>
<td>B.3 Asymmetric Cryptography</td>
<td>300</td>
</tr>
<tr>
<td>B.4 Hash Functions</td>
<td>302</td>
</tr>
<tr>
<td>B.5 Digital Signatures</td>
<td>303</td>
</tr>
<tr>
<td>B.6 Public-key Infrastructure (PKI)</td>
<td>305</td>
</tr>
<tr>
<td>B.6.1 Digital Certificate</td>
<td>305</td>
</tr>
<tr>
<td>B.6.2 Certification Authority (CA)</td>
<td>306</td>
</tr>
<tr>
<td>B.6.3 Certificate Lifecycles and Key Management</td>
<td>307</td>
</tr>
<tr>
<td>B.7 Individualization</td>
<td>308</td>
</tr>
<tr>
<td>B.8 Tamper resistance</td>
<td>309</td>
</tr>
</tbody>
</table>
C  Formal Verification Code: RDS Protocol 311
   C.1  The RDS Protocol Model 311
   C.2  ATL Formulae Used in The Verification of The RDS Protocol 323

Word Count: 79,151
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>State-of-the-art of the current License reselling proposals</td>
<td>95</td>
</tr>
<tr>
<td>4.1</td>
<td>Actions and consequences of delaying $Msg_2$</td>
<td>145</td>
</tr>
<tr>
<td>4.2</td>
<td>Exponentiation operations performed by protocol participants</td>
<td>151</td>
</tr>
<tr>
<td>4.3</td>
<td>Items and their size in each RDS protocol message</td>
<td>154</td>
</tr>
<tr>
<td>4.4</td>
<td>Communication cost of each RDS protocol message</td>
<td>154</td>
</tr>
<tr>
<td>4.5</td>
<td>Comparison between Chen’s protocol and the RDS protocol</td>
<td>155</td>
</tr>
<tr>
<td>5.1</td>
<td>Exponentiation operations performed by FSLRP participants</td>
<td>196</td>
</tr>
<tr>
<td>5.2</td>
<td>Comparison between our solution (FSLRP) and related work</td>
<td>199</td>
</tr>
<tr>
<td>6.1</td>
<td>Exp# performed when executing the RRP-MR and HC-MR methods</td>
<td>248</td>
</tr>
<tr>
<td>6.2</td>
<td>Comparison with related work</td>
<td>250</td>
</tr>
</tbody>
</table>
List of Figures

2.1 A typical DRM system architecture ........................................... 49
2.2 DRM fundamental principle: separation between license and content .......... 50
2.3 Dataflow of WM-DRM System ...................................................... 56
2.4 Dataflow of Apply FairPlay DRM .................................................. 59
2.5 OMA DRM Delivery Schemes of Version 1.0 [1] .................................. 62
2.6 OMA DRM V2.0 general architecture [2] ........................................... 64
3.1 License acquisition method in OMA-DRM [3] ..................................... 75
3.2 Consumer-to-consumer protocol of NPGCT system ................................ 84
3.3 Consumer-to-consumer protocol of Nuovo system .................................. 88
4.1 Zhou et al protocol with in-line TTP ................................................ 103
4.2 Zhou and Gollmanns protocol [4] with on-line TTP ............................... 105
4.3 Sokan and Shoup’s contract signing protocol with an off-line TTP .......... 107
4.4 Schematic figure of the RDS protocol .............................................. 116
4.5 Possible actions taken by Alice and Bob at each stage of a protocol ........... 117
4.6 General approach of formal methods .............................................. 123
4.7 MOCHA Verification Tool .............................................................. 126
4.8 A simple program written by the Game Guarded Command language ....... 129
4.9 RDS verification by MOCHA .......................................................... 131
4.10 A segment of the communication channel model ................................... 131
4.11 A segment of LI’s model .................................................................. 132
4.12 An example of guarded commands of A’s description ......................... 133
4.13 Part of guarded commands of B’s description .................................... 134
4.14 Part of guarded commands describing the player, \( Dis_A \) ...................... 134
4.15 Part of \( Dis_A \) description using guarded commands .......................... 135
4.16 RDS protocol verification using Mocha .............................................. 142
6.16 The verification process of LI’s signature on MRP ............... 237
6.17 The $M_{C_i}$ check ........................................ 238
6.18 The AKF check of the 2nd reselling ......................... 239
6.19 The verification process of $LA_i$ .......................... 241
6.20 The verification process of $LA_i$, where $i \geq 2$ .......... 242
6.21 LI’s verifications during the HC-MR method ............. 244
6.22 LI’s computational cost when one license is resold 5 times with the  
methods RRP-MR and HC-MR .............................. 249

A.1 Verification of $RP_{Lic}$ authenticity .......................... 283
A.2 $ASign_A$ signature generation on RD ...................... 284
A.3 Verification $BV_1$ .............................................. 285
A.4 Verification of $BV_5$ ........................................... 286
A.5 $ASign_B$ signature generation on $(RD||ASign_A)$ ...... 287
A.6 Verifying $ASign_B( )$ by Alice ............................. 288
A.7 Verification $LI_{V3}$ ............................................. 290
A.8 RPD hosted by LI .............................................. 291
A.9 The verification $AV_3$ .......................................... 295

B.1 The process of symmetric encryption .......................... 301
B.2 The process of asymmetric encryption ....................... 302
B.3 Digital signature generation and verification .................. 304
B.4 A digital certificate structure .................................. 307
Abstract

Consumers of digital contents are empowered with numerous technologies allowing them to produce perfect copies of these contents and distribute them around the world with little or no cost. To prevent illegal copying and distribution, a technology called Digital Rights Management (DRM) is developed. With this technology, consumers are allowed to access digital contents only if they have purchased the corresponding licenses from license issuers. The problem, however, is that those consumers are not allowed to resell their own licenses- a restriction that goes against the first-sale doctrine. Enabling a consumer to buy a digital license directly from another consumer and allowing the two consumers to fairly exchange the license for a payment are still an open issue in DRM research area.

This thesis investigates existing security solutions for achieving digital license reselling and analyses their strengths and weaknesses. The thesis then proposes a novel Reselling Deal Signing (RDS) protocol to achieve fairness in a license reselling. The idea of the protocol is to integrate the features of the concurrent signature scheme with functionalities of a License Issuer (LI). The security properties of this protocol is informally analysed and then formally verified using ATL logic and the model checker MOCHA. To assess its performance, a prototype of the RDS protocol has been developed and a comparison with related protocols has been conducted. The thesis also introduces two novel digital tokens a Reselling Permission (RP) token and a Multiple Reselling Permission (MRP) token. The RP and MRP tokens are used to show whether a given license is single and multiple resalable, respectively. Moreover, the thesis proposes two novel methods supporting fair and secure digital license reselling. The first method is the Reselling Deal (RD) method which allows a license to be resold once. This method makes use of the existing distribution infrastructure, RP, License Revocation List (LRL), and three protocols: RDS protocol RD Activation (RDA) protocol, and RD Completion (RDC) protocol. The second method is a Multiple License Reselling (MLR) method enabling one license to be resold N times by N consumers. The thesis presents two variants of the MLR method: RRP-MR (Repeated RP-based Multi-Reselling) and HC-MR (Hash Chain-based Multi-Reselling). The RRP-MR method is designed such that a buyer can choose to either continue or stop a multi-reselling of a license. Like the RD method, the RRP-MR method makes use of RP, LI, LRL, and the RDS, RDA, and RDC protocols to achieve fair and secure reselling. The HC-MR method allows multiple resellings while keeping the overhead on LI at a minimum level and enable a buyer to check how many times a license can be further resold. To do so, the HC-MR utilises MRP and the hash chain cryptographic primitive along with LRL, LI and the RDS, RDA and RDC protocols. The analysis and the evaluation of these three methods have been conducted.

While supporting the license reselling, the two methods are designed to prevent a
reseller from (1) continuing using a resold license, (2) reselling a non-resalable license, and (3) reselling one license a unauthorised number of times. In addition, they enable content owners of resold contents to trace a buyer who has violated any of the usage rights of a license bought from a reseller. Moreover, the methods enable a buyer to verify whether a license he is about to buy is legitimate for re-sale. Furthermore, the two methods support market power where a reseller can maximise his profit and a buyer can minimise his cost in a reselling process. In comparison with related works, our solution does not make use of any trusted hardware device, thus it is more cost-effective, while satisfying the interests of both resellers and buyers, and protecting the content owner’s rights.
Declaration

No portions of the work presented in this thesis have been submitted in support of an application for another degree or qualification of this or any other university or educational institute.
Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see [http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487](http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487)), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see [http://www.manchester.ac.uk/library/aboutus/regulations](http://www.manchester.ac.uk/library/aboutus/regulations)) and in The University’s policy on presentation of Theses.
Dedication

In the name of Allah

Thy Lord hath decreed, that ye worship none save Him, and (that ye show) kindness to parents. If one of them or both of them attain old age with thee, say not "Fie" unto them nor repulse them, but speak unto them a gracious word. (23) And lower unto them the wing of submission through mercy, and say: My Lord! Have mercy on them both as they did care for me when I was little. (24) [Al-Isra Chapter, The Holly Quran]

To those who have been always beside me, supporting me, and encouraging me for no personal benefits

To my father Mohammed and to my mother Horyia who passed away February 2010, may she be given mercy and peace.

I love you, I really do . . .

Tarek
Acknowledgement

First of all, I thank Allah (the lord) for all his blessings; I would not have completed my PhD without his guidance and success.

Secondly, I would thank many people who helped in different ways for this thesis to come true.

To my supervisor Dr. Ning Zhang for her dedication, guidance and valuable advice throughout my PhD. Her support, comments and feedback from the starting date until submission were invaluable and I really appreciate her effort.

To my advisor Dr. David Lester for this help and support. Also, to my colleagues, Rima, and Osama from IMG group. In particular, to Peter and Ali for their understanding, our valuable discussions, and for being on the same boat as me.

To Prof. Aboul Ella, for his endless help and support for more than 11 years. To Prof. H. Nassar, for all his kind help and encouragement, To Dr. Hatem, who is always beside me, for his inestimable help and support in different aspects. To my friends, Abdelmoneem, Dr. Mahmoud, Azab, M.Noor, and I.Achour for their various help and support. To my grandmother Hekmat and all her family for their endless prayers and help.

To my beloved mother Horyia who spent and sacrificed a lot of her life to see the moment of me getting a PhD but sadly, she passed away in 2010. To my father Mohammed who takes the most pride in me coming this far. I am very thankful for your love and endless support. To all my brothers, sisters, and uncles (especially Mr. Omar and Dr. Sayed) for being always helpful and supportive.

Last but definitely not least, I really want to thank my wife, Sally, for all the support and kindness that I have been overwhelmed with. Taking care of our kids, Yousif and Malik, has really made me concentrate on my studies and progress faster.

I would like to acknowledge the Suez Canal University, Ismailia, Egypt, for its financial support. I would like also to deeply thank the University of Manchester and the entire staff for all kind of help and support during pursuing my PhD.
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRM</td>
<td>Digital Rights Management</td>
</tr>
<tr>
<td>ATL</td>
<td>Alternating Temporal Logic</td>
</tr>
<tr>
<td>HC</td>
<td>Hash Chain</td>
</tr>
<tr>
<td>DSS</td>
<td>Digital Signature Standard</td>
</tr>
<tr>
<td>CRL</td>
<td>Certificate Revocation List</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>CA</td>
<td>Certificate Authority</td>
</tr>
<tr>
<td>SSL/TLS</td>
<td>Secure Socket Layer/Transport Layer Security protocol</td>
</tr>
<tr>
<td>TTP</td>
<td>Trusted Third Party</td>
</tr>
<tr>
<td>LRL</td>
<td>License Revocation List</td>
</tr>
<tr>
<td>REL</td>
<td>Rights Expression Language</td>
</tr>
<tr>
<td>WM-DRM</td>
<td>Windows Media DRM</td>
</tr>
<tr>
<td>OMA DRM</td>
<td>Open Mobile Alliance DRM</td>
</tr>
<tr>
<td>SMPTE</td>
<td>Motion Picture and Television Engineers</td>
</tr>
<tr>
<td>ODRL</td>
<td>Open Digital Rights Language</td>
</tr>
<tr>
<td>WMA</td>
<td>Windows Media Audio</td>
</tr>
<tr>
<td>WMV</td>
<td>Windows Media Video</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kits</td>
</tr>
<tr>
<td>GUID</td>
<td>Globally-Unique IDentity</td>
</tr>
<tr>
<td>SCMS</td>
<td>Serial Copy Management System</td>
</tr>
<tr>
<td>DAT</td>
<td>Digital Audio Tape</td>
</tr>
<tr>
<td>RD</td>
<td>Reselling Deal</td>
</tr>
<tr>
<td>CEDK</td>
<td>Content Encryption/Decryption Key</td>
</tr>
<tr>
<td>RP</td>
<td>Reselling Permission</td>
</tr>
<tr>
<td>SR</td>
<td>Single Resalable</td>
</tr>
<tr>
<td>MR</td>
<td>Multiple Resalable</td>
</tr>
<tr>
<td>MRP</td>
<td>Multi-Reselling Permission</td>
</tr>
<tr>
<td>AKF</td>
<td>Active Keystone Fix</td>
</tr>
<tr>
<td>RPD</td>
<td>Read-only Public Directory</td>
</tr>
<tr>
<td>RRP-MR</td>
<td>Repeated Reselling Permission-based Multi-Reselling</td>
</tr>
<tr>
<td>HC-MR</td>
<td>Hash Chain-based Multi-Reselling</td>
</tr>
</tbody>
</table>
Definitions

**Digital Rights Management (DRM):** A protection technology which refers to a set of hardware and software technologies and services which (1) control the authorised use of a given digital content, (2) and manage, through associated usage rights, any consequences of this use during the entire lifetime of the content.

**Authorised Domain:** A domain in which a group of devices are equally authorized to play/view digital contents, e.g. a home network where a number of devices are interconnected, so a digital content can be moved from device to device seamlessly.

**Fair license reselling:** A license reselling process such at the end of a reselling process undertaken between a reseller and a buyer, either the reseller receives the payment and the buyer receives the license, or neither of them receives anything useful

**Reselling permission:** A digital token showing that an associated license is resalable and can be resold within a particular period.

**Multiple Reselling Permission:** A digital token proving that an associated license is multiple resalable (i.e. authorising a license to be resold N-times).

**Non-resalable license:** A license which is issued without any reselling permission.

**Single resalable license:** A license that comes with a reselling permission showing that this license can only be resold once. When a buyer has bought this license from a reseller, it becomes a non-resalable one, so the buyer cannot resell it again.

**Multiple resalable license:** A license that is provided with a multi-reselling permission allowing the license to be resold multiple times (N-times).

**MR Type I (MR-I) License:** An MR License which can be resold N times by N different consumers. Suppose that a consumer, $C_1$, owning an N-time resalable license, can resell it to another consumer, $C_2$. Then, $C_2$ can also resell it to a third consumer, $C_3$, and so on, until the number of times this
license has been resold is N. Once a reseller (e.g. \( C_1 \) or \( C_2 \)) has resold this MR-I license once, he cannot reuse nor resell it again. In other words, for this type of licenses, a reseller can only resell it once. Every time the license is moved from one consumer to another, a reselling counter associated to this license is decremented by one till it reaches to zero at which point the license cannot longer be resold.

**MR Type (MR-II) License:** An MR license which can be resold \( N \) times by only one consumer, \( C_1 \). This means that (a) after each reselling of the license and provided that the number of times the license has been resold is less than \( N \) times, \( C_1 \) will still be able to use it on his device and able to resell it; (b) once this license is resold \( N \) times, \( C_1 \) can no longer resell or reuse it. There is one restriction with this MR-II type license. That is, any consumer, who has bought a copy of this MR-II license from another consumer, will not be able to resell it again.

**MR Type (MR-III) License:** An MR license which can be resold \( N \) times by \( M \) consumers where \( N \geq M \). In other words, a consumer, \( C_1 \), owning an \( N \)-time MR-III license, can resell it to another consumer, \( C_2 \) with the right for \( C_2 \) to resell this license again \( X \) times, where \( X < N \). Both of \( C_1 \) and \( C_2 \) can use and/or resell this license as long as the upper limit, \( N \) and \( X \), respectively, do not reach to zero. Once both of \( X \) and \( N \) become zero, the owner of the license can use it but can no longer resell it.

**Reselling Deal:** A contract to be agreed between a reseller and a buyer for a license reselling/purchase. It includes: (a) terms and conditions for this deal, (b) the price to be paid by the buyer, (c) the RD validity period, and (d) both Alice’s and Bob’s signatures.

**Keystone:** A random number that is used to bind ambiguous signatures to their respective signers.

**Keystone fix:** The hash value of a keystone.

**Active Keystone Fix:** A keystone fix of an MRP permission. It is needed in the latest reselling process of a license specified in the MRP. This keystone fix is a hash value of a random number called keystone. This keystone is created by LI and shared between LI and the authorised license reseller.

**License Revocation List:** A signed list containing all the revoked licenses issued by a particular LI.

**Pre-official reselling deal:** A reselling deal which is carrying both a reseller’s and a buyer’s ambiguous signatures.
### Notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>Alice (a Reseller).</td>
</tr>
<tr>
<td>$B$</td>
<td>Bob (a buyer).</td>
</tr>
<tr>
<td>$LI$</td>
<td>License Issuer.</td>
</tr>
<tr>
<td>$Lic$</td>
<td>The identity of the license to be resold, i.e. Alice’s license.</td>
</tr>
<tr>
<td>$RD$</td>
<td>A contract, called Reselling Deal (RD), that has been agreed by Alice and Bob.</td>
</tr>
<tr>
<td>$ks$</td>
<td>A random number called Keystone to be used in the concurrent signature.</td>
</tr>
<tr>
<td>$f$</td>
<td>a keystone fix that is a hash value of the keystone $ks$.</td>
</tr>
<tr>
<td>$RP_{Lic}$</td>
<td>A reselling permission for a license, $Lic$.</td>
</tr>
<tr>
<td>$MRP_{Lic}$</td>
<td>A Multiple Reselling Permission of a license, $Lic$.</td>
</tr>
<tr>
<td>$PK_i$</td>
<td>Public key of entity $i$.</td>
</tr>
<tr>
<td>$SK_i$</td>
<td>Private key of entity $i$.</td>
</tr>
<tr>
<td>$M</td>
<td></td>
</tr>
<tr>
<td>$E_{PK_i}$</td>
<td>Asymmetric encryption using entity $i$’s public key.</td>
</tr>
<tr>
<td>$ASign_i$</td>
<td>An ambiguous signature created by entity $i$.</td>
</tr>
<tr>
<td>$Sign_i$</td>
<td>A digital signature created by entity $i$.</td>
</tr>
<tr>
<td>$H()$</td>
<td>A cryptographic hash function.</td>
</tr>
<tr>
<td>$Lic$−$File$</td>
<td>The license file containing usage rights signed and granted by LI.</td>
</tr>
<tr>
<td>$RD_{DLA}$</td>
<td>A deadline for Alice to confirm that she has revoked her license.</td>
</tr>
<tr>
<td>$RD_{DLB}$</td>
<td>A deadline for Bob to activate a signed RD.</td>
</tr>
<tr>
<td>$E_K$</td>
<td>a symmetric encryption using a secret key, $K$.</td>
</tr>
<tr>
<td>$T_A$</td>
<td>A time read from the clock of Alice’s DRM client.</td>
</tr>
<tr>
<td>$RP_{period}$</td>
<td>A period within which a reselling permission, $RP_{Lic}$, is valid to be used in reselling the associated license, $Lic$.</td>
</tr>
<tr>
<td>$N_i$</td>
<td>A nonce which is created by entity, $i$.</td>
</tr>
<tr>
<td>$RD_{Pre-official}$</td>
<td>This is a pre-binding RD carrying both Alice’s and Bob’s ambiguous signatures.</td>
</tr>
<tr>
<td>$Install\text{-Status}$</td>
<td>The status of installing a up-to-sate LRL on Alice’s device.</td>
</tr>
<tr>
<td>$Payment_B$</td>
<td>The payment which Alice and Bob have agreed on in the negotiation phase. This is the amount that Bob should pay to LI to obtain $Lic$.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Introduction to Digital Rights Management (DRM)

With the vast use of the Internet and the technological improvements of media streaming and compression, digital contents (e.g. music, images, video, eBooks and games) can be instantaneously distributed over the Internet to content consumers. Nowadays many digital providers use these technology improvements to sell their digital contents over the Internet along with selling them through CDs. However, if these digital contents distribution is done without protection and management of digital rights, digital contents can be illegally and easily copied, altered, and distributed to a large number of recipients. Consequently, media companies could experience significant loss of revenue. A recent report [5] states that the revenue loss for 2007 due to music piracy for US recording companies reached to $ 5.3 billion. To protect the intellectual property of the digital contents and to avoid digital piracy, there is a need to prevent unauthorised access to digital contents and to manage usage rights of digital contents properly.

One of the most promising technologies to protect and manage digital contents is the Digital Rights Management (DRM) technology[7]. This technology offers a persistent content protection against unauthorised access to the digital contents. It also allows content owners to manage usage rights over their digital contents. Examples of these rights include copy permit, pay-per-view, one-week rental, etc.

\footnote{The SmartLM project [6] and the DRM system share the concept of managing licenses. The former manages licenses used in Grid environment while the latter manages licenses used in copyrighted digital contents.}
It only allows authorised consumers to access digital contents. The authorised
consumers are those who buy digital licenses of digital contents to be accessed.
Upon getting the licenses, they can access the contents according to the usage
rights granted in these licenses \[7\]. More detail about DRM is given in Chapter
2.

1.2 Introduction to Concurrent Signature Scheme

The concurrent signature scheme is a digital signature scheme where two entities
can initially exchange two non-binding signatures that are somehow linked to their
respective signers. With an additional piece of information, called a *keystone*,
being released either both signatures are binding to their true signers, or neither
is. The CS scheme was first introduced by Chen et al. at in \[8\]. This CS
scheme consists of four algorithms: (1) a SETUP algorithm used to establish
system parameter values necessary for creating and verifying signatures; (2) an
ASIGN algorithm used to create an ambiguous signature on a message; (3) an
AVERIFY algorithm used to verify an ambiguous signature produced by the
ASIGN algorithm; and (4) a VERIFY algorithm used to bind an ambiguous
signature to its rightful signer. These algorithms are described below.

**SETUP Algorithm:** The SETUP algorithm is a probabilistic algorithm,
which takes a security parameter as its input and outputs some public and pri-
ivate parameter values. For the public parameter values, the SETUP algorithm
randomly generates two large prime numbers, \( p \) and \( q \), such that \( q | (p - 1) \),
and creates a generator \( g \) of a multiplicative subgroup of order \( q \) in \( \mathbb{Z}_p^* \). In addi-
tion, it outputs two cryptographic hash functions, \( H_1 \) and \( H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q \).
The hash function, \( H_1 \), is only used for generating a keystone fix (i.e. a hash
value of the keystone \( ks \) ) , whereas \( H_2 \) is used to create other hash values to
be used for computing the required signatures. Moreover, the SETUP algorithm
defines descriptions of message space \( M \), signature space \( S \), keystone space \( K \),
and keystone fix space \( f \). These spaces are defined as follows: \( S \equiv F = \mathbb{Z}_q \)
and \( M \equiv K = \{0,1\}^* \). For the private parameter values, the SETUP algorithm
outputs a private key for each of the participants (two signers in our case). These
private keys, \( SK_i, i \in 1,2 \), are chosen uniformly at random from \( \mathbb{Z}_q \). The partic-
ipants’ public keys, \( PK_i, i \in 1,2 \), are then computed as \( PK_i = g^{SK_i}(mod p) \) and
they are also declared as public values.
**ASIGN Algorithm:** Once the system parameter values are established, the initial signer uses the ASIGN algorithm to ambiguously sign a message $m$. The ASIGN algorithm is a probabilistic one that takes $(PK_i, PK_j, SK_i, f, m)$ as its input and outputs an ambiguous signature, $ASign$, where $PK_i$ and $PK_j$ are two public keys $PK_i \neq PK_j$, $SK_i$ is the private key corresponding to $PK_i$, $f$ is equal to $H_1(ks)$, $ks$ is a keystone, and $m \in M$ is the message to be signed. Upon providing this input, ASIGN performs the following operations:

1. Pick a random number as the keystone $ks \in K$ and compute $f = H_1(ks)$, where $f \in F$;
2. Generate a random number, $r \in Z_q$; and
3. Calculate the following values:
   
   (a) $h = H_2(g^f PK_j^f (mod p)||m)$;
   
   (b) $h_1 = h - f (mod q)$;
   
   (c) $s = r - h_1 SK_i (mod q)$.

The output of the ASIGN algorithm is an ambiguous signature on $m$, denoted as $ASign = (s, h_1, f)$.

**AVERIFY Algorithm:** The ambiguous signature is verified using a AVERIFY algorithm. AVERIFY is a deterministic algorithm that takes the tuple, $(ASign, PK_i, PK_j, m)$, as its inputs, where $ASign = (s, h_1, f)$, $s \in S$, $h_1$ and $f \in F$, $PK_i$ and $PK_j$ are public keys, and $m \in M$ is the message signed. The algorithm then checks whether equation (1.1) holds. If it holds, AVERIFY outputs an accept. If the equation does not hold, AVERIFY outputs a reject.

\[
h_1 + f = H_2(g^s PK_i^{h_1} PK_j^f (mod p)||m)mod q \tag{1.1}
\]

**VERIFY Algorithm:** Once the keystone is released, a VERIFY algorithm is used to determine the originator and the recipient of a given ambiguous signature. The VERIFY algorithm is defined in terms of the keystone hash function $H_1(\cdot)$ and the AVERIFY algorithm. The VERIFY algorithm takes two inputs: $ks$ and $S_i$, where $ks$ is the keystone from which the keystone fix $f$ is computed, $S_i = (ASign, PK_i, PK_j, RD_i)$, and $ASign = (s, h_1, f)$ is an ambiguous signature on $m$. Given the inputs $(ks, S_i)$, the VERIFY algorithm first checks whether the hash value of the $ks$ is equal to the keystone fix $f$ used in creating $S_i$, (i.e., checks
whether \( H_1(ks) = f \). If they are not equal, the VERIFY outputs a \textit{reject} result. If they are equal, AVERIFY \((S_i)\) is executed. If AVERIFY \((S_i)\) outputs \textit{reject}, VERIFY likewise outputs \textit{reject}. If AVERIFY \((S_i)\) outputs \textit{accept}, VERIFY also outputs \textit{accept}. The pair \((ASign, ks)\) is called concurrent signature and ASign becomes binding to its respective signer. In other words, the VERIFY algorithm is one that is used to bind a given ambiguous signature to its actual signer/originator.

1.3 DRM and Consumers’ Rights

Although DRM enables content owners to protect the rights of their digital contents, it does not do the same with content consumers. It has been reported in [9] that current DRM systems give too much power to the rights’ holders, neglecting the rights of consumers. In the physical world, when music or other audio-visual materials are put onto records or cassettes, buying these materials is very simple. Consumers buy records or cassettes from content providers or distributors. The consumers, consequently, own these physical contents. They can then (1) play them on all their players, (2) lend them out to friends, (3) donate them to charity shops, or (4) even resell them to other consumers. In addition, they could copy them, but the quality of the copied content is not as good as the original. With these types of contents, consumers’ rights were relatively clear and could often be enforced.

Digital contents which can be put onto CDs, DVDs and other recording media, are very easy to copy and distribute. Digital content can be perfectly copied with nearly no effort and no cost. From this a problem arises where content authors, artists and creators cannot be sufficiently rewarded if their work can be freely copied and distributed/sold without restriction. In other words, in the digital world, there is a need for new rules and regulations to protect the rights of the owner of the original digital work. Access controls should be used to govern how digital contents may be used and persistent protection should remain with the digital content. This is achieved by a technology called Digital Rights Management (DRM).

The DRM technology, as reported in [10], [11], and [12], can provide integrity protection, and manage intellectual ownership throughout the whole value chain. It can also increase consumer choice between different business models such as
rental, and pay-per-view. With this DRM technology consumers who have paid for a digital content should be given the rights equivalent to those that are given to the buyers of a non-digital content (e.g. the right to resell a digital content). Sadly, the current DRM technologies allow consumers to practise very few rights over digital contents.

A report produced by the INDICARE project has pointed out a number of consumer issues that have not been addressed in current DRM solutions. These issues include the following points (for more detail, please read )

- **Access and usage control**: Consumers of digital contents should be entitled to the rights similar to those that are available to the consumers of the physical contents.

- **Fair use**: Any restrictions on digital contents should be in line with the consumer rights granted by copyright law.

- **Privacy**: If personal data has been collected using DRM technologies, consumers should know where their personal data are stored and who gathers the data, and who may use information about one’s actual consumption of digital content.

- **Interoperability**: The interoperability for consumers is described as the possibility of using digital contents in multiple ways and on multiple devices. DRM technologies should support this interoperability.

- **Software and hardware**: Almost all the current DRM systems require the installation of specific software or hardware. Therefore, consumer organisations, e.g. BEUC, have demanded that DRM systems must not limit the use of other protection software on consumers’ machines.

- **Pricing and product diversity**: DRM technologies should not establish monopolistic market structures and the prices of DRM-protected contents should be reflected upon potential usage restrictions.

We believe that a DRM system should provide a fair balance between the rights of content owners and the rights of consumers. Yes, it is a very difficult

---

2 The acronym INDICARE stands for Informed Dialogue about Consumer Acceptability of DRM Solutions in Europe. INDICARE focused on consumer issues with Digital Rights Management (DRM). While the project formally ended in February 2006, members of the project team are still working on the topics.
and protracted task to achieve this balance. Here, in this thesis, we will focus on one right (i.e. the reselling right) and propose a solution that could realise the reselling of a digital content in a fair and secure manner.

1.4 Research Motivation and Challenges

By investigating current DRM systems \cite{17, 18, 19} and as reported in \cite{13}, it can be concluded that current DRM systems mainly focus on the protection of content owners’ rights. They have not given due consideration to consumers’ rights. In particular, these systems do not permit consumers to resell the licenses they have purchased. Reselling something that a consumer rightfully owns (including digital licenses) is a legitimate right under the first-sale doctrine \cite{20}. This observation has motivated the research reported in this thesis.

Supporting license reselling is in the interest of consumers and content owners. A license reselling would give consumers a right to resell what they have already bought, so they could gain money from unwanted contents. It would also enable content owners to establish a new business model. In this model, licenses can be divided into two categories, non-resalable licenses, and resalable licenses. A resalable license may be set at a higher price than a non-resalable one. A consumer (i.e. reseller) in this model actually plays the role of a proxy for a content distributor helping content distributors to reach out to a broader market, increasing license distributions.

To support this legitimate right (i.e. reselling a digital license), there are three main challenging issues that need to be addressed:

1. **Content owner’s rights:** How to support a license reselling facility while protecting content owners’ rights;

2. **Fair license reselling:** How to make a reselling process fair to the buyer and the reseller;

3. **Reseller’s and buyer’s interests:** How to accommodate a reseller’s and a buyer’s monetary interests in a reselling process.

**Content owner’s rights:** With regard to protecting content owners’ rights in a reselling process, the following issues should be addressed: (1) the reseller must not be able to continue to use a license after it has been resold; (2) the
reseller should only be allowed to resell a resalable license, i.e. the reseller must not be allowed to resell a non-resalable license (we assume there are two types of license: resalable and non-resalable, and each of the resalable licenses has a reselling validity period); (3) a resalable license should only be allowed to be resold once; (4) a resalable license can only be resold within its reselling validity period. (5) a content owner must be able to trace a buyer of a resold license if this buyer has violated any of the usage rights of the license.

Fair license reselling: Fair license reselling process means that at the end of a reselling process undertaken between a reseller and a buyer, either the reseller receives the payment and the buyer receives the license, or neither of them receives anything useful. In addition to achieving this fairness property, two further issues have to be resolved in order to support fair license reselling. The first is the issue of DRM license transfer. When a license is originally purchased from a License Issuer (LI), the license is released to and bound to the original buyer’s (i.e. now the reseller’s) device. If this license is to be resold to a new buyer using the reselling facility, the license will need to be removed from the original buyer’s device, and installed on the new buyer’s device. However, this new buyer may not be able to access or operate this license after the license is moved onto his device. This is because during the license issuing process, this license was bound only to the old buyer’s device (i.e. the reseller’s device). If the new buyer has paid the reseller for the license, the buyer should be able to access the license received from the reseller. In other words, the reselling facility should allow a buyer to operate and use a second-hand license once he has paid for and acquired the license from the reseller. The second issue is the non-repudiation of engaging in a reselling process. There should be a way to prevent the reseller and the buyer from falsely claiming to have engaged in a reselling process or not. Without this protection, a reseller may have indeed resold his license but falsely claim he has not, thus attempting to resell the license more than once. If this happens, the content owner’s rights will be violated.

Reseller’s and buyer’s interests: In addition to the security and fairness issues discussed above, it would also be beneficial to accommodate both buyers’ and resellers’ monetary interests in a reselling process. Implementing the market power supporting both entities’ interests will make the license reselling facility much more attractive. A buyer’s interest, in this case, is to pay as little as possible for a second-hand license. A reseller’s interest, on the other hand, is to
maximise the price of the license as much as possible. The challenge here is how to address these two conflicting interests in the reselling process. Furthermore, while accommodating the buyer’s monetary interest, the buyer wants to be assured that the license he is about to purchase is a legitimate one. A license is said to be legitimate for re-sale if (1) it is resalable, (2) it is still within its reselling validity period, (3) it has not yet been resold.

The scope of this thesis is to find answers to these challenging issues by investigating and designing effective mechanisms to support a fair and secure solution to the problem of digital license reselling.

1.5 Research Aim and Objectives

The aim of this research is to investigate the state-of-the-art solutions to a license reselling problem in a DRM context and design a fair, secure and cost-effective solution supporting digital license reselling in the Internet environment. By embedding the fairness property in the reselling process, we aim not only at helping consumers to practise their rights (i.e. reselling their licenses) but also at helping all the involved entities (i.e. a reseller and a buyer) to overcome any mistrust or misbehaviour committed while engaging in a license reselling process over the Internet. To achieve this aim, the objectives of this research are as follows.

1. To identify, analyse and specify requirements for the design of a fair and secure license reselling solution.

2. To investigate and critically analyse the current state-of-the-art solutions to the license reselling problem for DRM-protected contents against the specified requirements.

3. To investigate, examine and design methods to address the weaknesses identified in the current state-of-the-art solutions.

4. To investigate, examine and design methods for achieving the fairness and non-repudiation properties in the license reselling process.

5. To design protocols that integrate the methods designed in (4) to support fair, secure and cost-effective license reselling in the Internet environment. These protocols are aimed at satisfying the requirements specified and overcoming the weaknesses and limitations seen in existing solutions.
6. To informally and formally verify the security strength of the designed security methods and protocols.

7. To evaluate the performance of the designed security methods and protocols.

1.6 Research Methodology

The research in this thesis has followed the following methodology.

**Literature Research:** The first task of this research was to study in-depth the related work in the literature. We started by investigating the characteristics of the current DRM systems and understanding its mechanism. Then a research was done to identify gaps in these systems. This led to choosing the license reselling problem to be the subject of the research of this thesis. The next point was to critically analyse the related works of this problem to identify weaknesses and limitations in existing solutions and the main cause of the problems. Based on the features of the existing solutions and our vision of supporting license reselling, requirements were specified. The related work was then analysed against these requirements. From this analysis, gaps were identified and hypotheses were proposed. This led to the area of supporting consumers’ rights in DRM by allowing fair and secure license reselling of DRM-protected contents. The literature review was carried on throughout the entire project period, as new work was published it was reviewed and necessary findings were taken into account.

**Theoretical Work:** Upon specifying the requirements in the literature research stage, solutions addressing these requirements were proposed and designed. The solution ideas were repeatedly refined by considering input from existing work. Other considerations (e.g. supporting market power) were given to make the proposed solutions attractive to all the entities involved. At the end of this stage, two novel methods consisted of three protocols were proposed. The first method allows reselling of a license once whereas the second method allows one license to be resold N times.

**Analysis:** The next stage was to analyse the security of the proposed solutions. The analysis included two methods: informal analysis and formal analysis. The informal analysis was first used to verify the proposed methods against the

---

3 The security, in this thesis, means that the designed methods support the properties of non-repudiation, fairness, and abuse-freeness.
security requirements and attacks identified from the problem and from the literature review. The formal analysis was then applied to provide a more in-depth and systematic analysis of the proposed methods. Mocha model checker [21] was used to carry out the formal analysis.

**Evaluation:** After the completion of the security analysis stage, the evaluation stage was done. The evaluation consisted of theoretical-based evaluation and prototype-based evaluation. In the theoretical one, the computational costs were calculated and compared with the most relevant work to show the merits of the proposed solutions over the related work. In the prototype-based evaluation, a prototype was first built using Java and then the execution time was measured to confirm the theoretical evaluation results. The prototype was also used to test the system against attacks identified in the analysis stage.

**Publish Results:** To report the results obtained from the research conducted, we published our research results in journals, conferences proceedings and book chapters. Section 1.7 gives a list of publications produced from the research in this thesis.

### 1.7 Novel Contributions and Publications

The research work presented in this thesis has led to the following novel contributions and achievements.

**Novel Contributions**

1. The design, analysis, and evaluation of the Reselling Deal Signing (RDS) protocol. This is a novel fair and abuse-free contract signing protocol allowing a reseller and a buyer to sign a negotiated contract, called Reselling Deal (RD).

   a. To the best of the authors’ knowledge, this is the first protocol that integrates the Concurrent Signature (CS) scheme [8] with what is already available in the existing license distribution infrastructure (i.e. LI) to support fair and abuse-free RD signing process in a license reselling process. With the use of this protocol, we are able to achieve a secure license reselling with strong fairness and abuse-freeness protections.

   b. The RDS protocol does not make use of a dedicated TTP to achieve fairness. Rather, it introduces some additional tasks into LI to ensure
fairness. In this way, the additional computational cost introduced as the result of introducing security can be kept at a minimum level.

c. The analysis of the security properties of this protocol is performed. This analysis involves: (1) informal analysis of the protocol against the security requirements and well-known and identified attacks; (2) formal verification of the security properties using the Alternating Temporal Logic (ATL) and the model checker MOCHA.

d. The evaluation and the performance of the RDS protocol are conducted. The evaluation is given in terms of the computational costs and these costs are compared with related work. The performance is evaluated based on an implemented prototype of the protocol. The security is also evaluated by testing the prototype against different attacks.

2. The design, analysis, and evaluation of the Reselling Deal (RD) method, a novel method supporting fair and secure digital license reselling. The RD method allows a license to be resold once while protecting content owner’s rights. To achieve fair and secure license reselling, this method makes use of (1) a novel Reselling Permission (RP) token which has enabled differentiation between a resealable license and a non-resalable license and allows a buyer to confirm that a license he is about to purchase is resalable, and (2) three protocols, 2M-RDS (2-Message Reselling Deal Signing) protocol, RDA (Reselling Deal Activation) protocol, and RDC (Reselling Deal Completion) protocol. The 2M-RDS protocol is first executed between a reseller and a buyer to sign a deal, RD. The RDA protocol is then run between the buyer and LI to activate RD. Finally, the RDC protocol is executed between LI and the reseller to revoke the resold license on the reseller’s device.

3. The design, analysis, and evaluation of two Multiple License Reselling (RRP-MR and HC-MR) methods, novel methods enabling one license to be resold N times by N consumers. The RRP-MR method enables buyers to choose to continue or to stop a multi-reselling process of a license. This method makes use of (1) the novel RP token, RPD (Read-only Public Directory) running by LI, and (2) three protocols, 2M-RDS protocol, RDA protocol, and RDC protocol. The RRP-MR method is a straightforward extension of the RD method but the former allows a buyer (if he pays extra fees to LI) to repeatedly obtain new reselling permission for a license being
resold. The buyer can then use this permission to resell the license once more. The multi-reselling process stops when the buyer does not request a new reselling permission. The RRP-MR method achieves the same level of security and fairness as the case of the RD method. The HC-MR method introduces two more features over the RRP-MR method, (1) reducing overhead imposed on LI by not issuing RP at each reselling of a license, and (2) allowing a buyer to verify how many times a license can be further resold prior to engaging in the license reselling. These two features are accomplished by making use of (1) a novel MRP (Multiple Reselling Permission) token which is generated and signed by LI to be used N times to enable a buyer to verify that a corresponding license can be further resold N times, (2) an online check of a Read-only Public Directory (RPD) updated by LI after each reselling. Once this MRP is generated, it can be used N times without having been signed by LI at each reselling, so reducing costs imposed on LI. Along with RPD, the MRP token allows a buyer to check the remaining number of resellings of a license. The HC-MR method, like both of the RD and RRP-MR methods, makes use of the protocols, 2M-RDS, RDA, and RDC to facilitate the multiple resellings, and LI to achieve fair and secure license reselling.

The proposed three methods, RD, RRP-MR, and HC-MR have the following features. (1) They do not require any additional trusted hardware. Instead, They make use of the existing distribution infrastructure (i.e. LI and DRM client). By doing so, we ensure that whatever is available at LI will also be made available in a license reselling process, thus keeping the cost low for consumers who are the most important entity in the value chain. (2) To the best of our knowledge, our solution is the first that supports market power in the context of reselling a digital license, a feature not seen in solutions proposed seen in the literature. This market power enables both buyers and resellers to maximise their respective monetary interests. In addition, our solution is the license reselling proposal that supports multiple resellings of a digital license. (3) our solution is the first piece of work that makes use of software mechanisms to support fairness, non-repudiation, and abuse-freeness in a license reselling process. This is achieved using LI along with the Reselling Dealing Signing (RDS) protocol. (4) With the use of the RD, RRP-MR, and HC-MR methods, content owners can establish a
new business model. In this model, the content owners can differentiate between non-resalable, single-resalable, and multi-resalable licenses. The multi-resalable licenses should be more expensive than the single-resalable ones which should be more expensive than the non-resalable ones. With this model, not only the resellers will be able to gain monetary benefits, but also the content owner will do. This is because, in our reselling solution, the resellers actually play the role of being proxies for the content distributors, which have the potential to increase market penetration, and license distribution.

**Published Papers**

Parts of the research work presented in this thesis have been published in the following journals, and conference proceedings.

**Journal papers**


**Conference proceedings papers**


**Book Chapter**

1.8 Thesis Structure

The structure of this thesis is organised as follows.

Chapter 2 gives an insight into the DRM technology, how a typical DRM system works, examples of current DRM systems, and open issues in the research area of DRM. Appendix B presents the background of the cryptographic concepts used in DRM systems and introduces the building blocks that will subsequently be used in the design of the methods and protocols. Chapter 3 investigates the state-of-the-art in the area of selling and reselling of digital licenses, identifies missed problems in the existing license reselling solutions, and describes ideas to address these problems.

Chapters 5, 6, and 7 present our contributions and results of the conducted research, which have been published in [22], [23], and [24]. Chapter 4 focuses on fair exchange (i.e. contract signing protocols). It outlines the security requirements of a contract signing protocol in a license reselling process, reviews and analyses existing research on signature exchange and presents the novel contract signing protocol (i.e. RDS protocol) that forms the basic components in our license reselling solution. It also presents the informal (analysis against requirements and identified attacks and threats) and formal analysis of the RDS protocol using the model checker Mocha. In addition, it introduces theoretical and prototype-based evaluation of the protocol. Based on this RDS protocol, Chapter 5 introduces our Fair and Secure License Reselling Protocol (FSLRP) suite which allows a digital license to be resold once. It also presents analysis and evaluation of the FSLRP suite. Based on the RDS protocol and the idea of the FSLRP suite, Chapter 6 introduces two methods allowing one license to be resold N times with N consumers. Chapter 6 additionally gives analysis and evaluation of the multi-reselling solution of digital licenses. Chapter 7 concludes the thesis and suggests future research.
Chapter 2

Digital Rights Management
Overview

2.1 Chapter Introduction

In the physical world, the physicality of contents helps in the rights management of the contents. In other words, it helps in the copyright protection of these contents. This physicality provides some barriers to unauthorised use of the contents. In the digital world, contents are in a digital form. Digital files can be copied and transmitted to many consumers without limitation. This ease of copying and transmission has caused many serious problems for content owners who want to maintain ownership right over their contents.

To address these problems, Digital Rights Management (DRM) technology has been developed. DRM is a technology that allows only authorised consumers to access digital contents. It also enables content owners to manage usage rights over these contents. The usage rights are a set of privileges the consumers can practice on a digital content. Examples of these rights are copy permit, pay-per-view, and one-week rental, etc.

This chapter gives an overview of DRM systems, and it is organised as follows. Section 2.2 explains what DRM is and Section 2.3 gives a historical overview of DRM. A fundamental principle of a DRM system is given in Section 2.4 whilst the DRM involved entities are introduced in Section 2.5. In Section 2.6, the DRM components are described. Section 2.7 then addresses how a DRM system works whereas Section 2.8 describes three exemplar DRM systems. A number of DRM open issues are discussed in Section 2.9. Finally, the chapter is summarised in
2.2 What is DRM

DRM refers to digital technologies, which enable legal distributions of digital contents (e.g. ringtones, songs, video clips) while enforcing usage rights specified by content owners of these contents. DRM also refers to a set of hardware and software technologies and services which (1) control the authorised use of a given digital content, (2) and manage, through associated usage rights, any consequences of this use during the entire lifetime of the content [25]. These hardware and software technologies include trusted hardware, encryption, digital signature, and individualisation (more details in Appendix B).

In a typical DRM system, as depicted in Figure 2.1, a content owner first encrypts a digital content with a key and defines usage rights over this content. The content owner then sends the encrypted content and the key along with the usage rights to a content provider and a license issuer (LI), respectively. When a consumer is interested in the content, he gets the encrypted form of this content from the content provider. However, the consumer cannot access this encrypted content until he pays LI for a corresponding license. This license contains the key by which the consumer can decrypt the encrypted content. It also contains the usage rights that control the usage of the content [25, 26, 27, 28].

2.3 DRM History

DRM technologies were initially developed for software copy protection in IT industries and for controlling the access of paid TV programs on set-top boxes in audiovisual consumer electronics. One of the earliest DRM systems was the Serial Copy Management System (SCMS). SCMS was developed in the mid-1980s to prevent illegal copies of the first generation of digital recording technology, Digital Audio Tape (DAT). SCMS is based on copy-control bits that can be set either to 00, “unrestricted digital copies allowed”, 11 “one generation of digital copies allowed”, and 10 “no digital copies allowed” [35].

DRM technologies have undergone two generations of development. The first has paid the full attention to security and encryption to prevent unauthorised copying. The second generation (i.e. the current one) has a broad aim. It covers
2.3. DRM HISTORY

Figure 2.1: A typical DRM system architecture

the protection, description, identification, trading, monitoring and tracking of all types of usage rights over digital contents.

2.3.1 First Generation DRM Systems

In the 1990s, InterTrust introduced the term “Digital Rights Management” (DRM) to describe technological means of restricting access to digital contents [35]. At that time, DRM systems were indented to prevent a consumer from making illegal copies and illegal distributions. The 1st generation of DRM systems used encryption techniques to bind digital contents to a single device. Consequently, a consumer could not share the content with his/her friends. He may make copies and distribute them, but receivers of these copies need a new decryption key. In other words, the receivers of the copied content have to contact and pay a license issuer to obtain the decryption key before they can play/view the content.

A well-known example of a DRM system representing this generation is DigiBox [29]. DigiBox is a protection technology that enables packaging of a digital content and its usage rights into a secure container so that this content can only be accessed according to the associated usage rights.

2.3.2 Second Generation DRM Systems

By the end of the 1990s, the Internet became increasingly popular and a cost-effective means for content distribution. This led to the issue of protecting digital
contents from illegal distributions becoming more urgent. Therefore, there was
a need to develop solutions to control the use of digital contents after their dis-
tribution. In addition to encrypting digital contents, the 2nd generation of DRM
systems has addressed the protection, description, identification, trading, moni-
toring and tracking of all types of usage rights over the digital content [30].

With the 2nd generation of DRM systems, consumers are allowed to view/play
their contents on more than one device, and to share their contents with their
family members and friends. This is achieved through a concept called Authorised
Domain. Examples of these DRM systems include OMA DRM [18] and Apple’s
FairPlay [19].

2.4 DRM Fundamental Principle

A basic concept underlying DRM systems is the separation between two files, i.e.,
a digital content and a license. As demonstrated in Figure [2.2] when trading non-
DRM contents, the content is the main asset. On the other hand, when trading
DRM contents, a license of a content is the main asset. In DRM systems, prior
to sending a content to a consumer, this content is encrypted using a symmetric
key known as Content Encryption/Decryption Key (CEDK). Once this content is
obtained by a consumer, it will remain protected. To access it, the consumer must
purchase a corresponding license. This license contains usage rights (explained
below) that are previously defined by an owner of this content.

![Figure 2.2: DRM fundamental principle: separation between license and content]

With the DRM technology, there are two modes for delivering the encrypted
content and its associated license: combined or separated. This feature allows a
flexibility of different business models for a content owner. For instance, a content owner may use Super-distribution business model [31]. In this model, a consumer can send his encrypted content to a second consumer, and the second consumer has to purchase a license from a license issuer before being able to access the content.

**A DRM-protected content**

A DRM-protected content is a digital file that contains an encrypted content object and other objects including:

- Unique identifier (i.e. Content ID) for this encrypted content object to link this content with its usage rights
- Information about the encryption algorithm used
- Information about LI of this DRM-protected content (e.g. LI’s URL and/or License Acquisition URL)
- Information describing the content, for example, the content’s title
- Content hash value (optional) to protect the integrity of the content.

**A License of a DRM-protected content**

A license is an XML-based file which is generated to describe usage rights for a DRM-protected content. A typical license file contains usage rights, CEDK, metadata, and LI’s signature. The usage rules are a set of rights that a consumer can practise with his digital content, such as copy permit, pay-per-view, a one-week rental, etc. *CEDK* is a symmetric key which is used to encrypt or decrypt the associated digital content. *A metadata file* contains the following fields, [2]:

- License ID: A serial number which uniquely identifies a license.
- Content ID: An identification number to uniquely identify a digital content.
- Content Provider ID: An identification number to uniquely identify a content provider.
- License Acquisition URL: an URL from which a license can be downloaded.
- Cryptographic Parameters: (Where needed/applicable)
• Hash of DRM-protected Content: This is to protect the integrity of the content. Note that this hash value will be included in LI’s signature, so if the content is modified, the signature will be invalid.

LI’s signature is a digital signature generated by LI on the license file to protect its integrity and prove its authenticity.

2.5 DRM System Entities

The implementation of DRM systems differs from system to system (see Section 2.8). The entities involved in a DRM system often have different names and use different ways to achieve their designated functions. However, the basic DRM process is the same, which usually involves four entities: a content owner, a content provider (or content distributor), a clearing house (or license issuer) and a consumer. In the OMA DRM system [3], only three DRM entities are considered a content provider, a clearing-house and a consumer. A content owner and a provider are held to be one entity. This entity is responsible for content encryption, packaging, defining usage rules and distribution. However, this thesis uses the former model, i.e. a content owner is separated from a content provider. This separation copes with the internet era. The content owner can be located in one country and the content provider in another far away country. Figure 2.1 shows the entities involved in a typical DRM system and the interaction among these entities.

**Content Owner**

A content owner is an entity that holds the ownership rights of a content and defines usage rules of the content. A content owner is responsible for (1) encrypting his content with a symmetric key called *Content Encryption/Decryption Key (CEDK)*, (2) creating metadata, including Content ID, License Acquisition URL, and encryption algorithm used, (3) sending this encrypted content to a content provider, (4) defining usage rights over this content, (5) sending these rights and the CEDK key to a license issuer. Examples of content owners are a film studio and e-publisher.

**Content Provider**

A content provider is an entity which provides consumers with encrypted content. It receives an encrypted content from a content owner, creates metadata
(e.g. to promote the content), and presents the content in web catalogues for the consumer.

**License Issuer**

A license issuer (LI) is an entity that is responsible for (1) authenticating a consumer (actually authenticating a consumer’s DRM client), (2) issuing a license for a given content to a consumer, (3) processing payments, (4) monitoring the consumption of a licence by a consumer. In addition, LI may also facilitate the reselling of a license (as in our case).

**Consumer**

A consumer is an entity that represents a person who wants to purchase a DRM-protected content and play/view it on his device. To do so, the consumer may need to install a DRM client and to buy a license associated to this content. The consumer could download this DRM client from a content provider’s website and then install it on his device. To buy the license, he needs to authenticate himself to an LI and then make a payment to LI through a clearinghouse. He can then get the license to play/view the content.

### 2.6 DRM System Components

As described in Section 2.5, a DRM system needs to perform different tasks during a content life-cycle. As described below, various components and tools are needed to achieve these tasks.

1. **Secure Containers**: They are used to make a digital content accessible only to authorised consumers (i.e. those who have paid). These containers can be achieved by using cryptographic techniques, e.g. DES [32] or AES [33]. Examples of these containers include InterTrusts DigiBox, and Microsoft eBook file format (.lit).

2. **Rights Expressions**: They are used to express digital rights (licenses) associated to a digital content in a machine-readable format. Rights expression languages mainly consist of rights, assets, and entity. DRM systems use Rights Expression Language (REL) to specify rights to contents. Currently, there is no uniform format for REL languages. Different REL languages for various applications have been developed by standardisation organisations. All these REL languages are typically based on eXtensible
Markup Language (XML). The main REL languages include ODRL (Open Digital Rights Language) [34], MPEG-21REL [35], and OMA (Open Mobile Alliance) DRM-REL [36]. These RELs have been widely used with multimedia contents, electronic publishing, mobile communication, and other fields.

3. **Content Identification and Description Systems**: They are used to (1) uniquely identify a digital content (e.g. International Standard Book Number), and (2) to associate contents with metadata describing them (e.g. SMPTE’s Metadata Dictionary\(^1\)).

4. **Identification of People and organisation**: Like digital contents, all entities involved in a DRM system need to be uniquely identified. For example, a consumer has to be uniquely identified before he is issued a license. These identification systems are very important for DRM systems as they only enable access of legitimate consumers to digital contents.

5. **Authentication Systems**: To support authorised access, DRM systems need techniques to authenticate legitimate consumers requesting access to digital contents. This can be accomplished by using cryptographic-based authentication methods (e.g. PKI). Typically, a Trusted Third Party (TTP) would be needed to issue digital certificates to all the involved entities. These certificates will then be used in an authentication process in a DRM system. Here are two examples:

   - LI has to authenticate a consumer’s DRM client, resident on a consumer’s device, before issuing a license to this client (i.e. to the consumer).
   - Different entities, involved in a DRM system, authenticate each other to establish an authenticated and secure channel between each pair of them.

6. **Watermarking and Fingerprinting**: These are technologies which persistently bind specific information (e.g. an identifier) to a digital content.

   In DRM context, watermarking and fingerprinting are typically used to help

\(^1\)SMPTE Metadata Dictionary has been introduced by the Society of Motion Picture and Television Engineers (SMPTE). It consists of a set of metadata items describing video/audio content: http://www.smpte-ra.org/mdd/.
2.7 HOW A DRM SYSTEM WORKS

a content owner to detect if there is a violation of their usage rights. This is why these technologies are known as forensic DRM technologies.

7. Reporting Events: In some business models, such as pay-per-view, a reporting event mechanism is required to enable event-based payments to proceed. This mechanism could also be used by organisations of collecting royalties.

8. Payment Systems: They are used to enable the monetary transactions between a consumer and a license issuer. Two types of payment systems can be used for making payments in DRM systems, (1) credit card/bank account, or (2) electronic cash.

2.7 How a DRM System Works

A typical DRM system works as follows. On a content owner’s side, (1) a digital content is symmetrically encrypted with a CEDK key, (2) a content metadata (e.g. Content ID, and License acquisition URL) is generated, (3) usage rights over the content are defined, (4) the content and its metadata are finally packaged through a packager. The packaged content is sent to a content provider, and the usage rights are delivered to a License Issuer (LI). When receiving the packaged content, the content provider prepares different methods (e.g. Website catalogue, CDs) to deliver the content to a consumer. Also, upon the receipt of the usage rights, LI puts them in the form of a license as discussed in Section 2.4. To do so, LI makes use of a Rights Expression Language (REL), described in Section 2.6.

When a consumer wants to purchase a particular DRM-protected content, he first uses his web browser to select and download this content from the web catalogue of the content provider. To access this content, the consumer needs first to install a DRM client on his device and then purchase a corresponding license from LI. To get the license, the consumer must (1) make a payment to a payment gateway (which is working in association with LI), (2) authenticate himself to LI’s system (i.e. the consumer’s DRM client authenticates itself to LI’s server). Upon successful payment and authentication, LI issues (i.e. generates and signs) a license to the consumer. LI then encrypts this license with the public key of the consumer’s DRM client and sends it to the consumer’s DRM client. After receiving the license, the client first verifies LI’s signature on the license,
ensuring that it is definitely issued by the authentic LI. The client then decrypts
the license using its private key and extracts CEDK. Finally, it uses the CEDK
to decrypt the encrypted content and passes it to a content render to play or view
it on the consumer’s device.

2.8 Existing DRM Systems

There are many commercial DRM systems on the market, e.g. FairPlay DRM [19],
WM-DRM [17], OMA DRM [18], and DReaM (developed by Sun Microsystems),
Intertrust Rights—system [37], RealNetworks’ Helix [38], and IBM’s Electronic
Media Management System (EMMS) [39]. In this section, we will give an overview
of the most successful DRM systems in the commercial market, i.e. WM-DRM,
FairPlay DRM, and OMA DRM.

2.8.1 Windows Media DRM

Windows Media DRM (WM-DRM) is a DRM system that is developed to support
a secure distribution of multimedia content based on the Windows Media Player
and Server. As illustrated in Figure 2.3, WM-DRM consists of four components
Windows Media Packager, License/Key Distribution Server, Content Distribution
System, and DRM Client.

![Figure 2.3: Dataflow of WM-DRM System](image)

Windows Media Packager is used by content owners to perform two tasks
(1) packaging a digital content in a particular format, and (2) setting usage and distribution rights for the content. In the packaging process, a content is first encoded to WMV or WMA (Windows Media Video/Audio), and then encrypted using CEDK key. A metadata about the content is also generated. This metadata includes:

- Content ID: It is a unique value, which identifies each digital content.

- License acquisition URL: It is the address of a web page from which a consumer can initiate a license acquisition process.

- Key ID: It is a string used by LI in the generation of the CEDK to be included in a license.

- Individualisation version number: It indicates that a consumer has to accept the Individualisation process. Otherwise, the content cannot be played/viewed. See Appendix B for more details about the Individualisation process.

- Other attributes: This field holds other information about the content such as artist, title, and content owner.

Once the content is packaged, it is sent to a Content Distribution System (i.e. content provider). Also, the usage rights and the key CEDK are sent to a License Server (i.e. LI). The Content Distribution System is the component that distributes the contents using various distribution methods (e.g. online shopping or CDs). The License Server is responsible for (1) distributing usage rights (i.e. licenses) and keys to consumers, (2) handling payment, and (3) authenticating consumers’ DRM clients.

The DRM client is the entity on a consumer’s device that enables the consumer to play/view a content. It first mutually authenticates a consumer to the License Server before the License Server issues a license to the consumer’s DRM client. It then acquires a content from the Content Distribution System and its associated license from the License Server. It finally allows the content to be played/viewed on a consumer’s device according to usage rights defined in the license.

**WM-DRM in the Market**: Different business models are supported by WM-DRM. These models include download, subscription, on-demand streaming, counted operations, and secure transfer of a protected content to portable devices.
As reported in [40], WM-DRM is adopted by large online music companies, e.g. Universal, Sony, EMI Music and many independent labels, to offer music tracks in digital format. Also many companies, e.g. BuyMusic, MusicMatch, Napster, MusicNow, DirectSong, MTVs URGE and Unbox, use WM-DRMs Windows Media Audio (WMA) format [41]. WM-DRM is additionally used by hardware device vendors such as Motorola, SanDisk, Philips and Toshiba [42].

**WM-DRM Advantages:** WM-DRM enjoys the following advantages. Firstly, it uses Windows operating system, whose media format is largely used over the Internet, and whose Media Player is implemented to support DRM technologies. Secondly, WM-DRM has got a flexible SDK (Software Development Kits) which can be used to design and implement various applications. Thirdly, it supports a flexible rights specification mechanism in which the content encryption and the rights specification are separate processes, supporting different business models. Finally, WM-DRM enables licenses to be transferred to mobile devices.

**WM-DRM Disadvantages:** The main disadvantages of WM-DRM system are: 1) it only supports two types of media format Windows Media Audio (WMA) and Windows Media Video (WMV) (i.e. Microsoft proprietary media formats); and 2) its DRM client is embedded into Microsoft Media Player for various devices, but does not support plug-in for other players.

### 2.8.2 FairPlay DRM

FairPlay DRM [19] is a protection technology developed by Apple to protect digital contents purchased from iTunes stores. It is built into a multimedia technology, known as *QuickTime*. It only works if Quicktime is installed on a consumer’s device. FairPlay also allows a consumer to play/view his content on five separate devices.

As illustrated in Figure 2.4, FairPlay DRM mainly consists of two components: an *iTunes store* and an *iTunes application* (i.e. *DRM client*). For the iTunes Store, a digital content is first encrypted using the AES algorithm with a key called *a master key*. The master key is stored in a ciphered form by encrypting it by another key known as *a user key*. This user key is generated when a consumer is purchasing a digital content from the iTunes store. When the consumer has

---

2Apple does not provide any information about FairPlay internal details. It also does not release any development kit for developing a compatible store or device. All the details in the literature are mainly taken from papers such as [43] [44] [45] [46].
bought the content, the iTunes store sends the consumer: the user key, the master key (encrypted by the user key), and the content encrypted by the master key.

To buy a track (i.e., digital content) from an i Tune store, a consumer first registers his device with the store by creating an account with the store. During this registration, the iTunes application creates a unique identifier for the device by hashing device identifiers such as drive name (C:), BIOS, CPU name and Windows ID. This unique identifier serves as the user key for this device. The user key along with the account information are stored on the iTunes store server. This registered device can later retrieve a user key to decrypt the corresponding master key.

On the consumer’s device, every time the consumer wants to play/view the content, the iTunes application, installed on the device, performs the following steps:

1. Uses the user key to decrypt the master key;
2. Uses the master key to decrypt the digital content;
3. Plays/views the content according to usage rights allowed.
CHAPTER 2. DIGITAL RIGHTS MANAGEMENT OVERVIEW

Note that, on the consumer’s device, the master key is stored as encrypted by the user key which is unique. Therefore, the master key cannot be transferred between devices. Also, this allows secure addition and removal of a device to and from the five registered devices.

As mentioned above, FairPlay permits a consumer to play/view his content on five devices. If the consumer wants to play his content on two devices (at home and at work), he has to authorise the new device with an iTunes store. This authorisation is performed by having the iTunes application send a unique machine identifier for the new device to the iTunes store. This device will then receive all the user keys corresponding to the account information. If the consumer has already registered five devices and wants to add a sixth device, he should de-authorise one device from the registered devices, and then authorise the new device. In the de-authorisation, the iTunes application will contact the iTunes server to remove the unique machine identifier and the user key of this device from the consumer database.

With authorisation and de-authorisation, Apple is able to restrict the number of devices to 5 authorised devices and ensures that each of these has got all the user keys required to play the digital content purchased from iTunes stores.

**FairPlay DRM in the Market:** FairPlay is mainly used by Apple products such as iTunes, iPhone, iPod, iPad and iTunes Store. The only exception is that Apple allowed Motorola mobile telephone company to use FairPlay DRM with the following three mobile phones sold by Motorola in the years 2005 and 2006: the Motorola ROKR E1, the Motorola RAZR V3i, and the Motorola SLVR L7.

**FairPlay DRM Advantages:** FairPlay DRM enjoys the following advantages. It allows consumers to play their content on up to 5 devices and on unlimited iPods; it also permits unlimited CD burnings but with quality degradation; it allows burning to CD (any single playlist of music can be burnt to CDs up to 7 times); and it enables a consumer to re-download an encrypted content from an iTunes Music Store.

**FairPlay DRM Disadvantages:** The disadvantages of the FairPlay DRM are that (1) it only supports two operating systems, Mac OS and Microsoft Windows, thus leaving Linux (as major operating systems) out of the usage, and (2) it only supports two types of digital format: MP4 (video) and AAC (audio).
2.8.3 Open Mobile Alliance (OMA) DRM

The Open Mobile Alliance (OMA) [18] is a standardisation organisation that develops open standards to enlarge the interoperability of mobile services. Members of OMA include device manufacturer members, e.g. Nokia, Sony Ericsson, Philips, Motorola, Samsung, and mobile operators (e.g. Vodafone, Orange, T-Mobile, LG Telecom). The DRM technology is one of OMA’s standardization activities. OMA DRM is an end-to-end technology developed to protect and distribute digital contents to devices. There are two versions of OMA DRM: OMA DRM V1.0 and OMA DRM V2.0.

OMA DRM V1.0

OMA DRM V1.0 was initially developed in 2002 to protect digital content such as video and audio clips, ring tones, wallpapers, and java games. OMA DRM V1.0 provides fundamental building blocks for a DRM system. As illustrated in Figure 2.5, it supports three simple protection schemes: forward-lock, separate delivery, and combined delivery [11].

- **Forward-Lock**: With this protection, a digital content cannot leave a device to which it was delivered. In other words, a consumer cannot transfer a DRM-protected content to another consumer’s device. Thus, Forward-lock supports a basic copy protection which protects content owners’ rights.

- **Combined Delivery**: This enables defining usage rights in addition to forward-lock. As shown in Figure 2.5(b), a DRM-protected content consists of two combined objects, a content and its usage rights (e.g. number of playing times and expiry date). With this feature, a content owner can create different kinds of business models, such as preview, and usage-based constraints.

- **Separate Delivery**: This is similar to the combined delivery except that the separate delivery adds extra security to the content. With this protection, a content is delivered in two separate files: one contains an encrypted content and the other contains usage rights for the content. The content is encrypted into DRM Content Format (DCF) by using a symmetric encryption technique and the usage rights and a symmetric Content Encryption Key (CEK) are put into a file called a license. The content and its license are delivered separately. This means that the content is useless without
its license. The separation between a content and its license supports the superdistribution business model. In this model, DRM protected content can be transferred from device to device but the receiver of the content has to acquire a new licence from a license issuer.

**OMA DRM V2.0**

OMA DRM V2.0 is designed to achieve a higher level of security and functionality than OMA DRM V1.0. Security, such as secure distribution, the authentication of devices, revocation and license integrity, are achieved through (1) providing mutual authentication between a Right Issuer (RI)\(^3\) and a consumer’s device, based on digital certificates, and (2) confidentiality and integrity protecting license based on public key encryption. The functionality is improved by introducing an authorised domain in which a consumer can have a limited number of devices and can access his content on all these devices\(^2\).

Based on the market feedback, two sub-versions (OMA DRM V2.1 and OMA DRM V2.2) of OMA DRM V2.0 have been released. OMA DRM V2.1\(^4\) was released in 2008. The main features of OMA DRM V2.1 over OMA DRM V2.0 include (a) metering: gathering information about the actual content usage, (b) RO installation confirmation, (c) content differentiation: with this mechanism a music track cannot be used as a ring-tone. OMA DRM V2.2\(^3\) was released

---

\(^3\)Known as License Issuer (LI) in our case.
in 2011 and it supports new features such as (1) the management of the advertisement based on the content acquisition and the content consumption, and (2) extended support of games and executables.

OMA DRM 2.0 consists of three major components a content issuer, a Rights Issuer (RI), and DRM agent.

**Content Issuer (CI)** is responsible for delivering a DRM content to consumers. CI first puts a content in a DCF format and then protects it using a symmetric Content Encryption Key (CEK). CI also includes additional data such as unique content ID and RI’s address into the packaged content. CI can deliver the DRM content to a DRM agent using different transport mechanisms, such as MMS (Multimedia Messaging Service) or Bluetooth. In addition, CI delivers CEKs to a Rights Issuer (RI).

**Rights Issuer** is an entity that specifies usage rights to DRM contents. RI may use Rights Expression Language (REL) to generate these rights. RI also generates licenses or Rights Object (RO) containing CEKs, received from CI, and these usage rights. This license is an XML file which expresses permission and constraints to be applied to the associated content. The license is then cryptographically bound to a DRM agent during its issuing process. Thus, the content can only be accessed with the appropriate license (i.e. RO).

**DRM agent** is a trusted entity which is hosted by a consumer’s device receiving a DRM content and its license. Each DRM agent is embedded with a public/private key pair and a digital certificate containing the public key. This certificate additionally contains information about the DRM agent such as issuer, device type, software version, and serial numbers. Based on this certificate, RI authenticates a given DRM agent before issuing and delivering it with a requested license. Furthermore, RI uses the agent’s public key to encrypt the license prior to sending it to this DRM agent. To access a given DRM content, the DRM agent opens the associated license and then renders the content according to the usage rights of this license.

If the DRM agent attempts to access a content but does not find the corresponding license, it sends a license request to RI. The request contains the device’s identity (i.e. its certificate) and the content ID obtained from the content package, DCF. Prior to issuing a license to this DRM agent, RI authenticates this DRM agent and makes sure that a financial transaction is performed. If positive, RI

---

4CI may receive the content pre-packaged from different sources.
issues a license containing usage rights and a CEK key which is further encrypted using a symmetric key called REK (Right-object Encryption Key). RI is then encrypting the REK key by the device public key, thus cryptographically binding this license to the target DRM agent. Therefore, only the target DRM agent containing the corresponding device private key can use the license to render the content.

OMA DRM in the Market: Producers of the most well-known mobile communication devices have agreed to adopt the OMA DRM V2.0. Examples of producers include Nokia, LG, Motorola, Samsung, Sony-Ericsson, and Siemens. Also, a number of mobile phone network operators are using OAM DRM 2.0. Vodafone, Orange, O2, and Cingular are examples of these operators.

OMA DRM Advantages: The main advantage of OMA DRM is that it has defined a standard for Mobile DRM which has allowed many mobile producers and operators to adopt it.

OMA DRM Disadvantages: The main disadvantage of OMA DRM is that it is only used with mobile phones. It does not support PCs.
2.9 Open Issues in Current DRM Systems

Consumers’ convenience and satisfaction with current DRM systems are still not addressed properly. Consumers’ privacy, DRM interoperability, first-sale doctrine (i.e. license reselling) are among open issues that need further research. In this section, we give an overview of these issues.

2.9.1 Consumer Privacy

DRM is a very helpful tool for content owners to deliver and distribute their digital contents, but it may affect consumers’ privacy. DRM systems do not give much attention to protecting consumers’ privacy. This is because consumers’ privacy is not in the direct interest of the content owners. To access a DRM-protected content, consumers are generally asked to disclose their identities to content providers or license issuers. Consumers may need to register with a content provider to either purchase content from the content provider (e.g. iTunes Stores), subscribe to read an eBook, or watch/play a content in the pay-per-view business model. Also, consumers’ identify information is collected to track the consumers in case of usage rights violation. For example, WM-DRM \cite{17} makes use of a globally-unique identifier (GUID) to track consumers \cite{19}. Also, FairPlay DRM uses a unique hardware identifier for the consumer’s device to limit the consumer to using the content on 5 computers only. Moreover, a client DRM installed on a consumer’s device can collect consumer’s playing statistics and send them to a License Issuer, who can use the statistics for different purposes, such as improving their services.

It is a challenging task to address these privacy problems. The main challenge to address the privacy problem in a DRM context is that of how a consumer can receive and play/view his digital content in an anonymous/pseudonymous way while the underlying security of DRM systems is not compromised. The literature contains a number of methods to address the consumer’s privacy problem and this includes:

- **Anonymity** - With this method, a consumer may use a content or service without his identity being revealed.

- **Pseudonymity** - It is a process with which a consumer can access a content or a service under a pseudonym. Thus, he does not need to disclose his real
identity.

- **Unlinkability** - It means that a sender and a receiver can communicate with each other without being identified as doing so.

- **TTP-based** - A Trusted Third Party can be involved to preserve a consumer's privacy in a DRM context.

Based on the anonymity and pseudonymity, Conrado et al in [50] have proposed a DRM system preserving consumer's privacy. A number of techniques, such as the unlinkability method [51] and blind signature [52, 53], blind decryption and hash chain [54], and anonymous cash [55], have been used to address privacy problems in a DRM context.

It is worth noting that the privacy concern is not only related to DRM-protected content, but also to DRM-free content. Recently, Apple iTunes stores launched a new service with which consumers can get a digital track without DRM restrictions. To get this type of content, a consumer has to pay a higher price, $1.29 instead of 99 cents. The consumer is then allowed to make many copies of the content he has bought and is permitted to play this content on different devices (i.e. iPod and others). However, within this DRM-free content, Apple iTunes have embedded the consumer's identifier information including the consumer's full name and account information including his e-mail address. This information is a great privacy concern to this consumer sharing the contents over p2p networks [56]. In general, consumers want to access digital contents in a way that their behaviour of consumption is not tracked nor profiled (i.e, using the content anonymously).

### 2.9.2 Interoperability

Interoperability is an important issue in the digital world. It can be noticed that computing devices used around the world are heterogeneous, but they access the Internet. With the DRM technology, it is also recommended that heterogeneous devices should have a similar level of accessibility with DRM-protected contents. As reported in [57], current DRM systems do not have a common DRM interoperability scheme. This is based on the fact that existing DRM systems make use of proprietary techniques (i.e. data formats, encryption, trusted hardware/software) in their system design. Without DRM interoperability, consumers need
to buy the same digital contents many times to be able to use them on their heterogeneous computer machines. A recent survey [13] has shown that European consumers would be happy to pay a higher price for digital contents with an interoperability feature.

The research literature contains several schemes [57, 58, 41, 59] to address the DRM interoperability problem. Generally, to achieve DRM interoperability, one of the following approaches could be used [58, 41]:

- **DRM Standardization**: This is the most obvious solution to the DRM interoperability problem, but it is the most difficult to achieve. As reported in [58], there are many challenges to achieving one DRM standardization. One of these challenges is that there would be little incentive for content owners to support for innovative business models, which are the ultimate goal for all involved entities in content distribution value chains, more importantly consumers.

- **Devices with Multiple DRM Systems**: With this approach, it is suggested that devices could be designed to support multiple DRM systems. However, this is an expensive approach as each device should support different hardware requirements, which are necessary for each DRM system.

- **DRM Translator**: In this approach, content and rights are translated from one system to another. The main challenge of this approach is the scalability issue. That is, as there are many DRM systems on the market, it is more difficult to find one DRM translator for all of them. This approach is used in [60, 57].

- **Connected Interoperability**: In this approach, as proposed in [59], an external TTP manages interoperability between different DRM systems. In this case, the TTP must to know all the security properties (such as encryption methods, and formats of both content and license) of involved DRM technology. However, it is very difficult for DRM providers to release their security properties to an external entity.

**2.9.3 First-sale: License Reselling**

Distribution of different types of items has individual focuses. In distributing a physical item like a book, the focus is on the controlled release of the physical
item subject to the required payment. However, in distributing a digital item such as a software or other digital content, the focus is on trading the license of the digital item. Thus, when a buyer wants to buy a digital content, he can first get the encrypted form of this content either from a content provider or from another consumer. Nonetheless, the buyer cannot access this content until he pays LI for the corresponding license. This license contains the key needed to decrypt the content. In this context, it is seen that if a consumer wants to resell the content, he actually resells the license of this content.

DRM follows the approach that usage rights, which are not specified in a license, are not permitted to consumers. For example, if you have bought a license allowing you to print 50 pages from an eBook, you cannot print 51 pages. Under this approach, consumers cannot resell their digital content as existing DRM systems do not allow consumers to do so in the granted licenses. It is believed that DRM solutions should be consistent with a first-sale doctrine in the sense that consumers should be allowed to resell what they have legitimately bought.

Current DRM systems, e.g. [17, 18, 19, 38], mainly focus on the protection of content owners’ rights. They have not given due consideration to consumers’ rights. In particular, these systems do not permit consumers to resell the licenses they have purchased. Reselling something that a consumer rightfully owns (including digital licenses) is a legitimate right under the first-sale doctrine [20]. To support this legitimate right, a number of license reselling proposals have been presented in the literature [61, 62, 50, 63, 64, 65]. These proposals fall into three classes depending on approaches used: full-trusted hardware, partial-trusted hardware, and non-trusted hardware. Proposals suggested in [61, 62, 63, 64] followed the full-trusted hardware approach. In this approach both a reseller and a buyer have to obtain a trusted device to perform any DRM-related process including reselling. On the other hand, solutions introduced in [50], and [65] followed the partial-trusted and non-trusted approach, respectively. A survey of all these approaches is given in chapter 3.

2.10 Chapter Summary

In this chapter, we have presented an introduction to DRM and identified the current open issues observed in DRM context. The chapter started by explaining
what a DRM system is and its history in protecting digital contents. It then showed that the separation between a content and a license is the main principle of a DRM system. This principle could enable various business models for content owners. In addition, the main entities involved and their roles in a DRM system are presented. Additionally, the chapter introduced the components of a DRM system and illustrated how this system works. Moreover, three well-known commercial DRM systems have been highlighted, and a number of open issues in the DRM research area have been discussed in this chapter. To understand how a DRM technology protects a digital content, the next chapter introduces the cryptographic mechanisms and other protection technologies used in the design of a DRM system.
Chapter 3

License Selling and Reselling solutions: A literature Survey

3.1 Chapter Introduction

This Chapter gives an overview of current solutions used to sell digital licenses in existing DRM systems and then discusses license reselling solutions proposed to support license reselling of a digital license. In detail, Section 3.2 presents an overview of the current license selling solutions being used with the existing DRM systems. A detailed survey on the proposed license reselling solutions is introduced in Section 3.3. In Section 3.4 we discuss what is missing in the current proposed solutions, while in Section 3.5 we introduce the best way to address the missing bits in the current reselling solutions. Finally, in Section 3.6 a summary of the chapter is given.

3.2 Current License Selling Solutions

Most of the current DRM systems are proprietary systems which make use of different approaches to secure and sell their protected contents. This leads to various selling methods which consumers have to use to purchase contents from different DRM systems. In this section, we give an overview of the license selling solutions used in the most well-known DRM system (i.e. WM-DRM (Window Media-DRM) [17], FairPlay DRM [19], and OMA (Open Mobile Alliance) DRM [18]).
3.2.1 License Selling Process in WM-DRM

License selling used in WM-DRM is supported with the use of a protocol known as a License Acquisition Protocol [66]. This protocol allows consumers to acquire a license using one of the two modes, silent or non-silent.

- **Silent license acquisition**: It is a process where a consumer has no interaction during a license acquisition. In this process, a consumer’s DRM client sends a license request to and receives a license response from LI without any involvement from the consumer. This type of license acquisition is only suitable when there is no need for any additional input from the consumer. For example, a consumer may have already registered and paid a subscription fee.

- **Non-silent license acquisition**: This is a process when a consumer is involved during a license acquiring process. During this process, the consumer may be asked to provide information (e.g. payment and/or username). This license acquisition method is typically used when a content owner wants to ensure that a consumer has checked particular information, e.g. terms and conditions, before issuing a license.

Here, we will describe the process of obtaining a license using a non-silent method [66]. Firstly, a DRM client installed on a consumer’s device creates a license request. This request contains **content header file and consumer’s system information**. The content header file contains **Content ID, License Acquisition URL, and key ID**

1, while the consumer’s system information contains **operating system type, DRM client ID, and client digital certificate**. Secondly, the DRM client signs the license request and then sends it to LI.

Once LI receives the license request, LI verifies the DRM client’s signature on the request. If the verification is positive, LI uses the information provided in the request to issue a license to the consumer. LI first searches in its database to find whether the content ID, provided in the license request, matches with any content ID in the database. If LI does not find matching, it then terminates the process. If LI finds matching, LI issues a license to the consumer by performing the following steps:

- Generate a license containing usage rules and CEDK key whose Key ID is provided in the license request.

1 Key ID is a string used by LI in the generation of the CEDK to be included in a license.
3.2. CURRENT LICENSE SELLING SOLUTIONS

- Sign the license to protect its integrity.
- Use the DRM client’s public key to encrypt the license.

Once the above operations are performed, LI sends the license to the consumer’s DRM client. Upon receiving the license, the DRM client verifies LI’s signature. If it is not valid, the DRM client declines the license response and terminates the process. If the signature is valid, the DRM client uses its private key to decrypt the license. It then uses the CEDK provided in the license to decrypt the content and allows the consumer to view/play the content.

3.2.2 License Selling Process in FairPlay

FairPlay DRM is designed such that usage rights (i.e. a license) are built into the DRM client (i.e. iTunes player) installed on a consumer’s device. Thus, the process of acquiring a content in FairPlay DRM corresponds to the process of acquiring a license in other DRM systems.2

To buy protected content (e.g. a song) through the iTunes store, a consumer must first install the iTunes media player which is embedded with a FairPlay DRM client. The consumer should then register with the iTunes store and then pay for the content through a credit card or other payment method (e.g. Paypal). In more details, a process of purchasing a content involves the following steps:

1. Download and install the iTunes media player.

2. Through the iTunes player, a consumer creates an iTunes account with Apple’s servers or iTunes stores. During the creation of this account, the device, e.g. PC or Mac running the player, will be authorised. In this authorisation, the iTunes player (1) generates a globally unique ID number for this device on, (2) sends this ID to Apple’s server which allocates this ID to the consumer’s iTunes account.

3. A consumer chooses a song from the iTunes music store and makes a purchase request to the iTunes server.

4. The iTunes player, installed on the consumer’s device, sends the purchase request along with the consumer’s system information to the iTunes server.

---

2As mentioned in Chapter 2, there are no technical details available about the FairPlay DRM, so the description of the content/license selling process in FairPlay DRM in this thesis is mainly taken from Wikipedia (en.wikipedia.org/wiki/FairPlay) and papers such as [43, 44, 45, 46].
5. A consumer uses his credit card or bank card to make a payment for the song.

6. After a successful payment, iTunes server generates a key called *master key* and uses this key to encrypt the song. The iTunes server then generates another key, known as a *consumer key* and uses it to encrypt the *master key*. It then sends the encrypted form of the song and the consumer key to the iTunes player.

7. Upon the receipt of these keys and the content, the iTunes player will use the consumer key to recover the master key. It will then use the master key to decrypt the encrypted content and view/play it according to the usage rights embedded in the iTunes player (i.e. DRM client).

### 3.2.3 OMA DRM License Selling

The process of selling or acquiring a license (or a Right Object as known in the context OMA DRM) in OAM DRM consists of two stages *Registration* and *License Acquisition* [2].

**Registration stage**: In this stage, a protocol called 4-pass Registration Protocol is executed between a consumer’s DRM client and LI (Right Issuer in OMA DRM context). This protocol is a handshaking and security information exchange protocol. Generally, it is only executed when a consumer contacts LI for the first time. However, if an update of the exchanged security information is required by LI, this protocol may be further executed. The registration protocol includes negotiation of the following items: protocol version, protocol parameters, device ID, LI ID, cryptographic algorithms, message integrity protection, and time synchronisation of the DRM client. Upon the successful completion of the protocol execution, the consumer can run the License acquisition protocol to obtain a license from LI.

**License acquisition stage**: In this stage, a protocol known as License Acquisition Protocol is executed between a consumer’s DRM client and LI to allow the client to acquire a license from LI. Upon successful completion of the protocol execution, the consumer’s device will obtain a license signed by LI and encrypted with the public key of the DRM client. As shown in Figure [3.1], this protocol

---

This protocol is known as RO Acquisition Protocol in the context of OMA DRM.
3.2. **CURRENT LICENSE SELLING SOLUTIONS**

consists of two messages[^1]: a **License Request** and a **License Response**

![Diagram of License Acquisition Method in OMA-DRM](image)

**Figure 3.1: License acquisition method in OMA-DRM**[^2]

**License Request:** To obtain a license from LI, a consumer’s DRM client sends a **License Request** message to this LI. This message consists of the following items:

- **Device ID:** This item is to identify the requesting device.
- **LI ID:** It identifies the requested LI.
- **Device Nonce:** It is a random nonce generated by the consumer’s device to counter a replay attack.
- **Request Time:** It is the current DRM Time as measured by the device.
- **License Info:** This item identifies the requested license.
- **Certificate Chain:** It is a certificate chain containing the device certificate.

[^1]: OCSP (Online Certificate Status Protocol) request and OCSP response messages, shown in Figure 3.1, are performed between Rights Issuer and OCSP Responder. In other words, they are not seen to DRM client. In addition, they are only executed if the license request contains a flag for OCSP. OCSP is an internet protocol to check the revocation status of an X.509 digital certificate.
information of the device certificate, the Certificate Chain item is not sent by the user.

- **Extensions:** A number of extensions can be included in the License Request which include:
  
  - **Peer Key Identifier:** This item indicates to LI that the LI’s public key has already been stored in the consumer’s device.
  
  - **No OCSP Response:** It tells LI that it is not required to send the OCSP response.
  
  - **OCSP Responder Key Identifier:** It identifies a trusted OCSP responder key which is stored in the consumer’s device.
  
  - **Transaction Identifier:** This item enables a device to provide LI with information for transactions tracking.

- **Signature:** This is the DRM client signature on the current message.

**License Response:** This is the second message of the license acquisition protocol. This message is sent from LI to the consumer’s device. Prior to sending this message to the device, as illustrated in Figure 3.1, LI may check the revocation status of the device certificate. Generally, the license response message contains the following items:

- **Request Status:** It shows whether the License Request has succeeded or failed.

- **Device ID:** This item is equal to the Device ID item sent by the device in the License Request message.

- **LI ID:** This item is the same value of LI ID item sent by the device in the License Request message.

- **DeviceNonce:** This is a nonce value which is equal to the Device Nonce sent by the device in License Request message.

- **Protected License:** This item contains usage rights and CEDK (Content Encryption/Decryption Key) which are encrypted using the device’s public key.
3.3 CURRENT LICENSE RESELLING SOLUTIONS

- **Certificate Chain:** This is a certificate chain including LI’s certificate. LI only sends this item the License request message does not contain the peer key identifier.

- **OCSP Response:** This is a complete set of valid OCSP responses.

- **Extensions:** An example of these extensions is Transaction Identifier that enables LI to provide the consumer’s device with information to track transactions.

- **Signature:** This is LI’s signature on the license response message.

For more information about the License acquisition protocol in OMA DRM system, see [3].

### 3.3 Current License Reselling Solutions

The literature contains a number of solutions [61, 62, 50, 63, 64, 65] proposed to address the license reselling problem. These solutions fall into three approaches: full-trusted hardware based, partial-trusted hardware based, and non-trusted hardware based. There are four solutions, [61, 62, 63, 64], which make use of the full-trusted hardware approach. As discussed in Section 3.3.1, these solutions require both a reseller and a buyer to obtain a trusted device to support digital license reselling. In the category of the partial-trusted hardware based approach, there is one solution [50]. This solution, discussed in Section 3.3.2 only utilises a smart card technology to perform the DRM related process. Finally, the solution presented in [65] makes use of the non-trusted hardware based approach. It does not require any trusted hardware in its design. This solution will be presented in Section 3.3.3.

Before giving a survey about the current license reselling solutions, we highlight the requirements for supporting fair license reselling in DRM systems. As discussed in Chapter 1, these requirements can fall into three categories:

1. **Protecting content owners’ rights:** This means that while supporting a license reselling facility, current owners’ rights should be preserved. These rights include:

   - **Preventing continued use:** A reseller must not be able to continue to use a license after it has been resold.
• Reselling resalable license: A reseller should only be allowed to resell a license if this license is resalable. We differentiate between two type of licenses: resalable and non-resalable. During the issuance of a license, LI will determine if the license involved is resalable or non-resalable and when this license can be resold (i.e LI can only allow a resalable license to be resold after one or two years from its release).

• Preventing unauthorised reselling: A reseller must not be allowed to resell a resalable license an unauthorised number of times.

• Buyer’s traceability: A content owner or LI must be able to trace a buyer of a resold (second-hand) license if this buyer has violated any of the usage rights of the license.

• Non-repudiation: A reseller, who has already resold his license, should not be able to falsely deny having resold this license.

2. Achieving fair license reselling: A fair license reselling means that at the end of a reselling process undertaken between a reseller and a buyer, either the reseller receives a payment and the buyer receives a license, or none of them receives anything useful.

3. Considering reseller’s and buyer’s interests: This includes monetary interest for both buyers and resellers, and other interests for a buyer. The monetary interest concerns maximising the benefits of both the buyer and the reseller. In other words:

• Allowing a buyer to pay as little as possible for a second-hand license.

• Allowing a reseller to maximise the price of the license as much as possible.

• Not adding additional cost to both a reseller and a buyer to resell and buy a second-hand license, respectively.

For the other interests for a buyer, the buyer should be assured that:

• License legitimacy: A license he is about to purchase is a legitimate one (i.e. it is issued by an authentic LI).

• Resale-ability check: A license he is about to buy is resalable and it is still within its reselling validity period.
3.3. CURRENT LICENSE RESELLING SOLUTIONS

3.3.1 Full-trusted Hardware based Solutions

There are four license reselling solutions \[62, 63, 64, 61\] making use of trusted hardware. With these solutions consumers (a reseller and a buyer) have to get special trusted devices to be able to play/view and resell his digital content/license. More details are given in the following sections.

3.3.1.1 Kwok’s DRM System

Kwok work \[61\] was the first solution to address the license reselling problem. The work proposed a DRM system (hereafter referred to as Kwok’s system) to support license purchase and resale. With this system, a consumer could use the system to acquire/purchase a license from a license issuer (LI) and to resell it later to another consumer.

Kwok’s system consists of two services: a DRM External Service Centre (ESC) and a Local Service Centre (LSC). The ESC service is located on LI’s side, and it is used for issuing a license to a consumer. The LSC service is located on a consumer’s device, and it is used for acquiring and using a license. ESC and LSC each has a public/private key pair.

To acquire a license from LI, a consumer, Alice, uses her LSC as follows. She first registers with LI’s ESC and makes a payment for the license to LI. Upon the successful completion of these steps, LI encrypts the license, called an official license, with Alice’s (i.e., LSC’s) public key and sends it to Alice. The LSC service, located at Alice’s device, receives the encrypted license from LI and decrypts it with its private key. It then stores the decrypted form of the license in a database provided by LSC.

To resell her license, Lic\(_A\), Alice first searches for a buyer, Bob. She then asks Bob to register with her LSC, LSC\(_A\). Bob then uses his LSC, LSC\(_B\) to register as a peer-consumer of Alice. Bob then makes a payment for Lic\(_A\) to Alice. When Alice receives the payment, she uses her LSC\(_A\) to generate a license called a peer-license. LSC\(_A\) then encrypts the peer-license using LSC\(_B\)’s public key and sends it to LSC\(_B\) on Bob’s device. Once the peer-license is sent to Bob, Alice’s official license will be encrypted in a decrypted form. This means that a reseller could resell this license to many consumers and could keep continuing using it as well, thus violating content owners’ rights.

\(^5\)In \[61\], Kwok et al did not mention that their system is designed under the assumption that the reseller’s device should be trusted. However, we believe that without making use of this assumption, the underlying security of the DRM system will be compromised. This is because the authors mentioned that the official license will be stored in consumer’s device in a decrypted form. This means that a reseller could resell this license to many consumers and could keep continuing using it as well, thus violating content owners’ rights.
License is disabled on her device by using her LSC, i.e., Alice can no longer use it. Also, upon receiving the peer-license, Bob can use it on his device.

**Limitations of Kwok’s system**

Kwok’s system was the first DRM system allowing a consumer to resell his license to another consumer. However, it only discussed the reselling problem from an abstract level. It did not give technical details on how a reselling is achieved and how content owners’ rights will be kept protected during and after the reselling. In other words, it fails to address the following problems.

- *Preventing continued use:* Although it has been mentioned that a reseller will not be able to reuse her license after its reselling, Kwok’s system fails to give technical details as how the reseller is prevented to continue using a resold license.

- *Preventing unauthorised reselling:* Kwok’s system does not address how a consumer is prevented to resell his license unauthorised number of times.

- *Consumer Traceability:* As described above, LI is not involved in a reselling process which is only performed between a reseller and a buyer. As a result, LI cannot obtain the buyer’s identity that allows LI to trace the buyer if she/he violates any of the usage rights of the license.

- *License legitimacy:* This is not discussed in Kwok’s system description.

- *Fairness:* Kwok’s system also does not have a mechanism to support fairness in the reselling process. As shown above, a reseller receives a buyer’s payment before sending the license to the buyer. In this case, the reseller could prematurely abort the protocol execution once he receives the payment. As a result, the buyer could be left in a disadvantageous position (i.e., made his payment without getting the license).

### 3.3.1.2 Sun’s System

In 2005, Sun et al. [62] introduced another DRM system supporting license reselling. Sun’s system also makes use of trusted hardware (trusted device and smart card) to enable playing/viewing and reselling digital licenses. This system permits a consumer to (1) acquire a license from LI, (2) play/view a content corresponding to this license through an online service provided by LI, (3) and resell the license to another consumer through LI and a *Second-hand Store (SS).*
To play/view a content, Alice first gets a digital content from a content provider. She then registers with LI. In this registration, Alice’s smart card sends LI Alice’s ID and content ID. Alice then makes payment to LI for a request license. Alice then uses LI’s online service to play/view the content. That is, Alice first authenticates herself, and makes the payment to LI via her smart card before obtaining a license corresponding to the content. With the license and the enclosed key, Alice can play/view the content. Each access to the content requires Alice to repeat the authentication process to get the key needed to decrypt the content. When Alice finishes playing/viewing the content, the trusted device is instructed to delete the license and the decrypted content. Thus, Alice can not abuse the decrypted content.

To resell her license $L_{icA}$, Alice makes use of a third party called SS acting as TTP between LI, Alice, and Bob. Alice first sends SS a reselling request containing content ID, Alice’s ID, and Alice private key. To accept this request, SS first checks with LI whether Alice is the legitimate owner of $L_{icA}$. If Alice is not the owner of $L_{icA}$, the reselling operation will be terminated. Otherwise, SS publishes on its website that $L_{icA}$ is for sale. When a buyer is interested in $L_{icA}$ posted on SS’s website, he sends SS a purchase request containing a payment for $L_{icA}$ and Bob’s identity. SS then requests LI to update the ownership of $L_{icA}$ such that Bob becomes the new owner the license (i.e. $L_{icA}$ will become $L_{icB}$). This means that Alice can not use $L_{icA}$ any more. Once the update has taken place and SS has received an acknowledgement with this update from LI, SS then sends the license, $L_{icB}$ to Bob. SS also sends the payment to Alice, thus achieving transfer (reselling) of $L_{icA}$ ownership from Alice to Bob.

**Advantages of Sun’s System**

It can be noticed that Sun’s system has addressed a number of limitations identified in Kwok’s one. Sun’s system has provided solutions to the following problems:

- **Preventing continued use**: Sun’s system only allows a reseller, Alice, to use a license through LI’s online services. Thus, once Alice has resold $L_{icA}$ and then she attempts to reuse it again, LI can prevent Alice from using this license.

- **Buyer traceability**: Since LI (and SS) has to receive Bob’s identity during the reselling process, Bob’s identity can be traced by LI.
CHAPTER 3. A LITERATURE SURVEY

• **License Legitimacy:** As discussed above, prior to accepting a reselling request from Alice to resell Lic\(_A\), SS checks with LI whether Lic\(_A\) is a license issued by LI. Hence, the license legitimacy is checked before permitting the reselling process.

• **Preventing unauthorised reselling:** It can be noticed that a reselling process is only performed through SS and LI. This allows LI to detect any unauthorised reselling of a license.

**Limitations of Sun’s System**
The limitations of Sun’s system are given below.

• **Using SS:** Adding SS in a reselling process makes it inefficient as SS adds more communication messages to the reselling process. In addition, SS must be unconditionally trusted by Alice, Bob and LI. Finding such an entity could be very difficult in the Internet-based DRM environment. If SS misuses any information obtained during the reselling process, SS can compromise the whole process. For example, SS can maliciously send LI the **content ID** and *Alice’s ID*, which are received from Alice in the reselling request, to play/view Alice’s content.

• **Fair reselling:** This has not been addressed in Sun’s system.

• **Adding cost to consumers:** As shown above, Sun’s system makes use of trusted hardware to prevent a consumer from keeping the license after the playing/viewing session is finished. It also utilises the smart cards to authenticate the consumer to LI and SS during playing content and reselling the license, respectively. The trusted hardware and the smart card usually introduce an additional cost to the consumers.

3.3.1.3 NPGCT DRM System

In 2005, Nair et al [63] proposed a trusted hardware based DRM system. This system is later called NPGCT\(^6\). NPGCT system allows a consumer not only to buy rights (license) from LI to access a specific content, but also to resell N-copies of this license to other consumers (buyers). To support this reselling while

\(^6\)NPGCT is an abbreviation for the authors of this system. These authors are Nair, Popescu, Gamage, Crispo, and Tanenbaum
avoiding the problems seen in Kwok’s and Sun’s system, NPGCT system heavily relies on the use of trusted devices.

Here we give an overview of these trusted devices. They are tamper-proof devices that only execute their embedded certified rules. These devices are also able to locally perform atomic actions. The atomic actions are actions that can be logically coupled such that either all or none of them is executed. The trusted devices provide two main features. Firstly, once a copy of a license is resold, a reseller’s device, $D_R$, automatically updates the original license to reflect how many copies of the license are left, i.e. how many copies the reseller can further resell. As a result, the reseller can not resell his license more than the legitimate $N$ copies authorised by LI in the original license. Secondly, once $D_R$ receives a payment order from buyer’s device $D_B$, $D_R$ performs two automatic coupled actions (i.e. atomic actions): (1) updating the license to reflect that one copy of it has been resold; (2) storing the payment order received from $D_B$. The atomicity of these two actions is very critical in the reselling process as it ensures that $D_R$ can not store the payment order without simultaneously updating the license indicating that one copy has been resold.

The license reselling process of NPGCT is as follows. Suppose that a consumer, Alice, has got a license, $Lic_A$, from LI. In addition, Alice has been authorised to resell $5$-copies of this license. To resell a copy, $Lic_1A$, of $Lic_A$, Alice follows a protocol known as Consumer-to-Consumer (C2C) protocol. Before executing the C2C protocol, Alice first searches for a buyer, Bob, to buy $Lic_1A$. Alice and Bob then negotiate a price for $Lic_1A$.

Once they agree on a payment amount, they allow their relative trusted devices to execute the C2C protocol. As illustrated in Figure 3.2, Bob’s device, $D_B$, first sends Alice’s device, $D_A$, a purchase request for $Lic_1A$. Then $D_A$ and $D_B$ mutually authenticate each other. In this authentication, $D_A$ and $D_B$ exchange and verify each other’s public key certified by CA (Certificate Authority). Upon valid authentication, $D_A$ generates a symmetric key, $K_{AB}$, and uses it to encrypt the content, $M$, associated to $Lic_1A$. It also signs $Lic_1A$ with its private key. $D_A$ then generates and sends $D_B$ a message containing $Lic_1A$ signed by $D_A$, $Lic_A$ received from LI, $M$ encrypted by $K_{AB}$, and the key, $K_{AB}$. Once $D_B$ receives this message, it performs the following verifications: (1) it verifies that the original right, $R$, and the content, $M$, are indeed issued by LI, i.e. it verifies LI’s signature on $Lic_A$, and (2) it verifies that $R'$ is generated by $D_A$, it verifies $D_A$’s signature
CHAPTER 3. A LITERATURE SURVEY

1-DB → DA: Buying request for a content, M
2-DB ← DA: Mutual authentication between DA & DB
3-DA → DB: \{E\{M\}_k'||E\{K'\}_PKDB\||R||R'||Lic_A||Lic_1_A||Mmt\}
   3.1 DB verifies Lic_1_A, R and M
4-DB → DA: { (payment) & AkgDB}

Protocol Notations

DA : Alice's trusted device
DB : Bob's trusted device
M : Content being resold
R : The original rights granted by Li to DA
R' : Rights granted by DA to DB
K' : Symmetric key generated by DA
PKDB : Public key of DB
Lic_1_A : License generated by Li and
          L = \{PK_{Li}||PK_{Da}||M||Mmt||R||\text{Sig}_{SK_{Li}}(PK_{Li}||PK_{Da}||M||Mmt||R)\}
Lic_A : License generated by DA and
        L' = \{PK_{Da}||PK_{Db}||M||Mmt||R'||\text{Sig}_{SK_{Da}}(PK_{Da}||PK_{Db}||M||Mmt||R')\}
AkgDB : Acknowledgement that DB has received M and its L' where
        AkgDB = \{PK_{Db}||PK_{Li}||E\{M\}_i||Mmt||R'||\text{Sig}_{SK_{Li}}(PK_{Da}||PK_{Db}||E\{M\}_i||Mmt||R')\}
Mmt : Metadata about M, e.g. artist name, and album and song title

Figure 3.2: Consumer-to-consumer protocol of NPGCT system

on Lic_1_A; (3) it checks that R' can be derived from R, (4) and it checks Mmt to make sure it matches the requested content. If these verifications are positive, DB sends DA the payment agreed prior to starting the C2C protocol. Also, DB sends DA sends an acknowledgement, AkgDB, which serves as evidence of receipt of the license Lic_1_A.

The purpose of sending Lic_A and Lic_1_A is to form a chain of licenses. This chain serves the same purpose which Lic_A serves for DA. In addition, the chain proves that DB has been issued rights, R', with respect to the content, M. If R' is authorising further redistribution, DB can use the chain, Lic_A and Lic_1_A, to do so in the same manner as DA has used Lic_A in the redistribution protocol.

Advantages of NPGCT system: NPGCT system enables a consumer to resell N-copies of his original license and during reselling these copies, it is successful in addressing the following issues:

- **Second-hand store:** A license reselling process is performed without using a second-hand store. This is achieved by using the trusted device. Not using this store reduces the communication overhead.

- **Li:** Even though NPGCT system does not use Li during a reselling process,
thanks to the trusted devices which enable NPGCT system to do so, performing a reselling process without LI further reduces the communication overhead of the system as there is no communication message between LI, and Alice or Bob.

• *Preventing unauthorised reselling:* This is also achieved by using the trusted devices. As explained above, with these devices, a license can only be resold \( \text{N} \) times specified in the original license.

• *License legitimacy:* Furthermore, the trusted device has helped NPGCT system to accomplish license legitimacy checks. As described above, prior to sending a payment to Alice, \( D_B \) verifies \( D_A \)'s signature on \( \text{Lic}_1A \) and LI's signature on \( \text{Lic}_A \) sent along with \( \text{Lic}_1A \). If these two verifications are positive, it means that \( \text{Lic}_1A \) is legitimately derived from a legitimate license issued by LI (i.e. \( \text{Lic}_A \)). As result, \( \text{Lic}_1A \) is legitimate.

**Limitations of NPGCT System**

The limitations of the NPGCT DRM system can be summarised as follows.

• *Adding cost to consumers:* NPGCT system makes use of special hardware devices (i.e., trusted devices), this introduces an additional cost into the underlying reselling process (i.e. adding an additional cost to consumers). This makes NPGCT system less competitive in terms of cost-effectiveness.

• *Reselling the original license:* NPGCT does not address reselling of original licenses. The main aim of the NPGCT solution is to allow a consumer to obtain a main license with the right to resell \( \text{N} \)-copies of the license. The solution has not considered the scenario and the implications of a reseller wanting to resell the main license once he has resold the \( \text{N} \)-copies of the license. These implications include (1) how to prevent a reseller from reselling the original license an unauthorised number of times, (2) how to prevent a reseller from continuing using a resold license.

• *Fair Reselling:* The NPGCT system, likes Kwok’s and Sun’s systems, has failed to address fair reselling of a digital license. As discussed above, Bob sends Alice the payment after he receives the license, \( \text{Lic}_1A \), from Alice. Thus, it is possible for Bob to obtain \( \text{Lic}_1A \) and then prematurely aborts the reselling process. This leaves Alice in a disadvantaged position.
• *Buyer traceability:* Only in one case, NPGCT system is able to trace a buyer, Bob. This case takes place when LI is involved in the payment issues (i.e. Bob has to pay Alice’s payment to LI). In this case, to collect this payment deposited by Bob, Alice has to report to LI the transaction taking place between her and Bob. However, as specified in the description of NPGCT system, Alice and Bob can choose another payment method. If this is the case, LI can not know Bob’s identity to trace Bob if he does violate usage rights of the license received from Alice.
3.3. CURRENT LICENSE RESELLING SOLUTIONS

3.3.1.4 Nuovo DRM System

In 2007, Jonker et al [64] identified two weaknesses in NPGCT system: replay attack and unfair reselling. They then designed the Nuovo system to address these weaknesses while allowing consumers to resell N-copies of a digital license to other consumers. In other words, the Nuovo system is considered as an extension of NPGCT.

As reported in [64], the NPGCT system is subject to replay attacks because the authentication step of the C2C protocol, shown in Figure 3.2, does not include any replay attack countermeasures. As a result, after the authentication phase, a malicious buyer could obtain two or more licenses from a previous session before sending payment for one license to a reseller. To overcome the replay attack, as depicted in Figure 3.3, the Nuovo system uses a nonce to ensure freshness of the entire exchange.

As discussed in Section 3.3.1.3, the NPGCT system does not provide a fair reselling as Bob could receive Alice’s license and then prematurely abort the C2C protocol without sending payment to Alice. To achieve fairness, Nuovo system makes use of LI. However, LI is only involved if there is a dispute (e.g. hardware failure) between Alice and Bob during a reselling process. In other words, Nuovo-C2C protocol comprises two sub-protocols: main-C2C and recovery-C2C.

Main-C2C protocol: The main-C2C protocol is executed between Alice and Bob. As shown in Figure 3.3, they first let their devices $D_A$ and $D_B$ to run a reselling process. If the reselling is successfully executed (i.e. if Alice receives Bob’s payment and Bob gets Alice’s license), the main-C2C protocol is considered as successful. If there is any problem (e.g. hardware failure) occurred during the protocol execution, LI is invoked to resolve this problem by launching the recovery-C2C protocol.

The recovery-C2C protocol: If Alice receives Bob’s payment at step 3 of the main-C2C protocol, and Bob does not receive $Lic_A$ at step 4, Bob will invoke the recovery-C2C protocol with LI. As illustrated in Figure 3.3, in the first two steps of the recovery-C2C protocol, Bob’s device and LI authenticate each other. In step 3, Bob’s device sends LI a recovery request. This request contains the items of the message sent to Alice’s device in step 3 of the main-C2C protocol. This is to prove that Bob’s device has sent Alice’s device the payment. LI then responds with $Lic_A$ to Bob’s device.

7 In Nuovo system, a content provider is acting as LI and a content provider.
Figure 3.3: Consumer-to-consumer protocol of Nuovo system

Advantages of Nuovo System

Like the NPGCT system, the use of trusted devices enable Nuovo system to achieve the properties: *preventing an unauthorised reselling, checking a license legitimacy, and not involving Second-hand Store*. In addition, Nuovo system has achieved a fair reselling by making use of LI along with the trusted devices.

Limitations of Nuovo System

Like the NPGCT system, the Nuovo system fails to address the problems of
adding cost to consumers, reselling the original license, and buyer traceability (see Section 3.3.1.3 for more details).

3.3.2 Partial-trusted Hardware Based Solution: Conrado’s System

There is one system designed using partial-trusted hardware (i.e. smart card). In 2004, Conrado et al. [50] proposed a DRM system that supports reselling digital license. Conrado’s system only makes use of a smart card technology as trusted hardware. Each smart card is embedded with public/private key pair. The public is known to the owner of the card and the private key cannot leave the card. It also contains a compliance certificate issued by a compliance certificate issuer for the smart card (CA-SC). This certificate indicates the validity of a smart card. Moreover, the smart cared contains a revocation list corresponding to the smart card public key. This revocation list contains a list of licenses, which have been issued to the smart card and have been revoked for some reasons, e.g. resold ones. This list must be periodically updated from CA-SC to contain any new revoked licenses.

In Conrado’s system, a smart card performs two operations. Firstly, It uses its public key for authentication operations (e.g. authenticate a buyer’s identity to LI when acquiring a license). Secondly, the smart card uses its secret key to perform any secure operation, i.e. decrypting the license received from by LI.

With the use of Conrado’s system a reselling process consists of four main steps: (1) getting an anonymous license, (2) reselling the anonymous license, (3) personalising the anonymous license, and (4) confirming license revocation.

Getting an anonymous license: In order for a reseller, Alice, to resell her license, $Lic_A$, Alice sends LI a request to resell $Lic_A$. Prior to allowing Alice to resell $Lic_A$, LI authenticates that $Lic_A$ was indeed issued to Alice (i.e., confirm whether Alice is a legitimate owner of $Lic_A$). If the authentication is valid, LI then marks $Lic_A$ as revoked and sends Alice an anonymous license $Lic_{Anon}$. LI additionally sends CA-SC a message indicating that $Lic_A$ has been revoked.

Reselling the anonymous license: Once Alice obtains $Lic_{Anon}$, she searches for a buyer, Bob, to purchase this license. When Alice and Bob agree on a price for $Lic_{Anon}$, they exchange $Lic_{Anon}$ for the price (i.e. the payment). Note that, at

---

8An anonymous license is a license which is not associated to any consumer yet and can only be used if it is personalised by LI.
this stage, though Alice has already resold her $\text{Lic}_{\text{Anon}}$ to Bob, Alice is still able to use the resold license, i.e. $\text{Lic}_A$, until she renews the compliance certification of her smart card from CA-SC.

**Personalising the anonymous license:** Upon receiving $\text{Lic}_{\text{Anon}}$ from Alice, Bob connects LI to activate this $\text{Lic}_{\text{Anon}}$ (i.e. personalise it to be associated to Bob). Bob first authenticates himself to LI and then sends $\text{Lic}_{\text{Anon}}$ to LI which will associate it to Bob’s identity (i.e. his smart card public key). LI then sends it to Bob who can then use it to access the corresponding content.

**Confirming license revocation:** In this step, LI will ensure that Alice can not continue using $\text{Lic}_A$. As mentioned above, when LI issues Alice $\text{Lic}_{\text{Anon}}$, LI informs CA-SC that $\text{Lic}_A$ has been marked as revoked. When the next time Alice renews the compliance certificate of her smart card, CS-SC adds $\text{Lic}_A$ to the revocation list embedded in Alice’s certificate. Once the new certificate is received by Alice’s device, it will be stored in Alice’s smart card. Whenever Alice attempts to use $\text{Lic}_A$, the revocation list is checked against the identity of $\text{Lic}_A$. If it is in the list, the smart card does not decrypt the license. Thus, Alice can not continue to use the license once it has been resold.

**Advantages of Conrado’s System**

The main advantages of Conrado’s system are that it enables a consumer to resell his license to another consumer. While supporting license reselling, Conrado’s system has addressed the following issues:

- **No Second-hand store:** As shown above, a license reselling process is performed without any intervention form a second-hand store.

- **License legitimacy:** As explained above, to resell $\text{Lic}_A$, Alice has to present $\text{Lic}_A$ to LI who checks whether $\text{Lic}_A$ is indeed issued by LI to Alice (i.e. $\text{Lic}_A$ is legitimate).

- **Unauthorised reselling:** As Alice has to contact LI to resell $\text{Lic}_A$, if LI has indeed allowed reselling $\text{Lic}_A$, LI can then detect any unauthorised reselling of $\text{Lic}_A$. LI can then prevent this reselling.

- **Buyer’s traceability:** With Conrado’s system, a buyer’s identity can be traced. As Bob must contact LI to personalise the anonymous license, $\text{Lic}_{\text{Anon}}$, received from Alice, LI is able to get Bob’s identity. LI then is able to trace this identity if Bob has violated content owners’ rights provided in $\text{Lic}_B$. 
Limitations of Conrado’s System
It does not address the following issues.

- *Preventing continued use:* Conrado’s system provides a mechanism by which a reseller is prevented from continuing using his resold license. As described above, this is done by making use of a license revocation list embedded in a smart card. However, this mechanism fails to prevent a continued use of a resold license once it has been resold. Consider the following scenario. Alice has asked LI to resell Lic\textsubscript{A}. When LI issues Alice Lic\textsubscript{Anon}, LI only marks Lic\textsubscript{A} as revoked. LI does not indeed revoke Lic\textsubscript{A} from Alice’s device. In other words, Alice will get Lic\textsubscript{Anon} and will continue to use Lic\textsubscript{A}. At this stage, there is no problem for LI or the content owner as only Alice can use Lic\textsubscript{A}. Now suppose that Alice has just renewed the certificate of her smart card (i.e. the revocation list is just updated). In addition, suppose that the next renewal is due after 60 days. Also suppose that one day after Alice has renewed her certificate, she has resold her Lic\textsubscript{Anon} to Bob. This means that to prevent Alice from using Lic\textsubscript{A} on her device, Alice should wait 59 days during which she can continue to use her Lic\textsubscript{A}. On the other side, once Bob has activated Lic\textsubscript{Anon}, he will be able to use the activated license (call it Bob’s license (or Lic\textsubscript{B} for short) which is equal to Lic\textsubscript{A}). This means that one license can be used on two different devices (Alice’s and Bob’s device) for 59 days. As a result, content owners’ rights could be violated.

- *Fair reselling:* As it can be seen above, Conrado’s system does not address the problem of fairness when Alice and Bob exchange Lic\textsubscript{Anon} for a payment, respectively.

- *Adding cost to consumers:* Conrado’s system makes use of the smart card technology which imposes an additional cost on the consumers. Such cost could hinder this system from wide adoption.

3.3.3 Non-trusted Hardware based Solution: Laila’s System
Under this category, there is one solution which is recently introduced by Laila et al [65]. Laila proposed a scheme called *Reselling Digital Content Scheme*
This RDCS enables a consumer to buy a digital license embedded into its content either from a content provider or from another consumer.

The RDCS scheme consists of four entities, an artist, a Trusted Authority (TA), a buyer, and a reseller. An artist is a person who has created a digital content and wants to sell it to consumers. TA (LI in our case) is an entity that (1) helps an artist to sell his contents, (2) assists a reseller to resell his content, and (3) tracks a dishonest consumer. A buyer is a consumer who wants to purchase a content either from an artist or from a consumer (i.e. a reseller). A reseller is a consumer who has bought a content from an artist and wants to resell it to another consumer. Each of these entities has got a public/private key pair.

The RDCS scheme works as follows. An artist, ART, first registers his content, $C$, with TA. In this registration, TA stores information including content ID, $ID_C$, and the artist’s public key. TA then publicises this information on a website. When a buyer, Alice, wants to buy $C$ from ART, Alice sends ART her public key, $PK_A$. ART then sends Alice $C$ and $Ownership_{CA}$ proving that Alice is the owner of $C$. This ownership is of the form $Ownership_{CA} = \{C||PK_A||Sign_{SK_{ART}}(C||PK_A)\}$, where $SK_{ART}$ is the artist’s private key. To play $C$, Alice has to let her device contact TA to register that Alice is the current owner of $C$ (i.e. TA adds Alice’s identity to $C$’s information). Also, whenever Alice wants to play $C$, her device must send a playing request to TA to check that Alice is still the owner of $C$.

To resell the content, $C$, to a consumer, Bob, Alice first needs to contact TA to agree on a public/private key pair called a proxy key pair (i.e. $PK_P/SK_P$). Alice will use it to produce one-time proxy signature during the reselling process. TA also updates $C$’s information reflecting that Alice has got a proxy key pair to resell $C$. Alice then uses $SK_P$ to create a new ownership for $C$, $Ownership_{CB}$, proving that Bob is the current owner of $C$. This ownership is of the form $Ownership_{CB} = \{C||PK_B||Sign_{SK_P}(C||PK_B)\}$ where $PK_B$ is Bob’s public key. Alice then sends Bob $C$ and $Ownership_{CB}$. Note, prior to engaging in buying $C$ from Alice, Bob can check whether Alice is the legitimate owner of $C$. This is achieved by checking AT’s website to verify whether $PK_A$ is included $C$’s record. When Bob receives $C$, and its $Ownership_{CB}$ from Alice, he first contacts TA to register that he is the current owner of $C$ (i.e. Alice can not play/view or resell

\(^9\)The author of RDCS mentioned that RDCS could be implemented using either trusted hardware or software. In this thesis, we will analyse it as a software-based solution.
3.3. CURRENT LICENSE RESELLING SOLUTIONS

$C$ again as $C$’s ownership is transferred to Bob). Bob can then play/view the content, $C$.

**Advantages of Laila’s System**

While supporting a license reselling, Laila’s system has also addressed the following issues:

- *No Second-hand store*: A license can be resold without making use of a second-hand store. Laila’s system only makes use of LI to enable Alice to resell $C$ to Bob.

- *License legitimacy*: This is achieved by making use of TA’s website. Prior to reselling a content, $C$, Bob can verify that $C$ is registered at TA’s website (i.e. it has an artist and a current owner).

- *Unauthorised reselling*: This is done as TA is involved in a reselling process. To resell a content, $C$, Alice has to get a proxy key pair from TA. If TA has indeed issued a proxy key pair for $C$, LI can then decline to issue another key pair, thus preventing any unauthorised reselling of $C$.

- *Preventing continued use*: As a content has to be played/viewed and resold with TA’s help, and TA can prevent a reseller from continuing using a resold license.

- *Buyer’s traceability*: As explained above, when Bob has got $C$ from Alice, he must contact TA to register that he is the current owner of $C$. Thus, TA is able to get Bob’s identity. TA is then able to trace this identity if Bob has violated the content owners’ rights.

**Limitations of Laila’s proposal**

Laila’s system fails to address the following issues:

- *Fair reselling*: It does not address the problem of fairness in the reselling process between a reseller and a buyer.

- *Limited applications*: As reported by the author, the RDCS scheme does not consider the mass market of music tracks and video clips, rather it only supports limited scenarios. It is designed to support content reselling for a piece of digital art (e.g, photographs or compositions of music) which could only be shown in private collections, like showrooms or museums.
museum, for example, could buy artwork to show it for a period of time and then want to resell it to another museum.

- **Online use**: Laila’s system only supports online use of a digital content. A consumer has to contact TA whenever he wants to play/view the digital content. This is not convenient for consumers.

- **Adding cost to consumers**: Laila’s system adds an additional cost to consumers if it is implemented using trusted hardware but this additional cost can be avoided if it is implemented using software.

### 3.4 What is Missing?

This section shows the findings of our survey of the current license reselling solutions. These findings are summarised in Table 3.1. From this table and from the discussion in Section 3.3, the following remarks can be drawn:

- None of the current reselling solutions has addressed the fairness property while supporting original license reselling. A license reselling process is a type of exchange process. A reseller exchanges his licenser for a payment from a buyer. At the end of this exchange, it is important for both reseller and buyer to either get their relative item or none of them get anything useful. Note that, Nuovo system has achieved fairness while reselling a copy of the main license, but, as explained earlier, this system does not address the scenario where the main license is being resold.

- None of these solutions have considered the monetary interests of both reseller and buyer. Implementing the market power supporting both buyers’ and resellers’ interests would make the license reselling facility much more attractive. A buyer’s interest, in this case, is to pay as little as possible for a second-hand license. A reseller’s interest, on the other hand, is to maximise the price of the license as much as possible.

- None of the existing reselling solutions have supported multi-reselling\(^{10}\). This means that an original license can be resold N times with N consumers

\(^{10}\)Nuovo system supports multi-reselling from a different prospective. It allows a consumer to resell N copies of his license while the consumer can continue using the main license. As discussed earlier, it does not consider a scenario where the consumer wants to resell the main license.
### 3.4. WHAT IS MISSING?

| Table 3.1: State-of-the-art of the current License reselling proposals |
|--------------------------|----------------|----------------|----------------|----------------|----------------|----------------|
|                          | Kwok | Sun | NPGCT | Nuovo | Conrad | Laila |
| Use second-hand store    | No   | Yes | No    | No    | No     | No   |
| Use license issuer       | No   | Yes | No    | Yes   | Yes    | Yes  |
| Play content: Online     | No   | Yes | No    | No    | No     | Yes  |
| Use trusted hardware     | Yes  | Yes | Yes   | Yes   | Yes    | Yes/No |
| Prevent continued use    | Yes* | Yes | No    | No    | Yes    | Yes  |
| Prevent unauthorised reselling | No   | Yes | No    | No    | Yes    | Yes  |
| License legitimacy       | No   | Yes | Yes   | Yes   | Yes    | Yes  |
| Buyer traceability       | No   | Yes | Yes   | No    | Yes    | Yes  |
| Resell original license  | Yes  | Yes | No    | No    | Yes    | Yes  |
| Support fair reselling   | No   | No  | No    | Yes** | No     | No   |
| Add cost to consumers    | Yes  | Yes | Yes   | Yes   | Yes    | No/Yes* |
| Support non-repudiation  | No   | No  | No    | Yes** | No     | No   |
| Resale-ability check     | No   | No  | No    | No    | No     | No   |
| Support monetary interest| No   | No  | No    | No    | No     | No   |
| Support multi-reselling  | No   | No  | No    | No    | No     | No   |

* Kwok only mentioned that resold license will be disabled on reseller’s device but no technical details are given
** Fairness and non-repudiation are only achieved with reselling copies of an original license
† Yes, if Laila’s system is implemented with trusted hardware, and No if not
++ Yes, if LI is involved in the payment process. Otherwise, No

where a reseller should not be able to resell nor continue to use a license once he has resold it. In other words, a consumer, Alice, can resell a license to another consumer, Bob. Once this is done, Alice should not be able either resell it again or continue using it and Bob should be able to resell the license again to a third consumer, Charlie. Bob then must not be allowed to resell nor continue use the license. This process is repeated until the license is resold N times.

- None of these solutions have achieved non-repudiation property in a license reselling process. Non-repudiation of reselling is very crucial for a content owner as it provides this owner with evidence that a reseller has indeed resold his license. Without such evidence, the reseller could indeed have resold his license but falsely claim having resold it, requesting to resell the
license again, thus violating the content owners’ rights.

- All the current solutions make use of trusted hardware, thus imposing an additional cost on their consumers. The only exception of this is Laila’s solution which could be implemented by software mechanisms. However, this solution has limitations. As reported in [65], Laila’s solution only considers digital artwork, e.g. photographs or digital imagery, which shall only be displayed in some private collections or museums (i.e. does not support tracks and video contents such as iTunes). In addition, with this solution, the consumers have to contact trusted party to play/view their digital contents (i.e. it is not convenient for consumers).

- None of the existing solutions differentiate between resalable and non-resalable licenses. This differentiation allows content owners to establish a new business model in which a resalable license would be more expensive than a non-resalable one.

- As there is no differentiation between a resalable and non-resalable license, current solutions do not consider a resalability check. This check enables a buyer to verify whether the license is about to buy is resalable.

The following section highlights our vision of supporting fair license reselling while preserving the underlying security of a DRM system.

### 3.5 The Best Way Forward

This section introduces the ideas used to design a license reselling facility while addressing the missing issues discussed above. These ideas are:

- Taking a more flexible approach to digital license reselling such that different types of digital license can be accommodated in a single framework in a DRM system. These type include (1) a single resalable license that comes with a digital token (reselling permission) showing that this license can only be resold once; (2) a multi-reselling license that is provided with a multi-reselling permission allowing the license to be resold multiple times (N times); (3) a non-resalable license that is issued without any reselling permission. In this way, different types of licenses can be supported, which,
3.6. CHAPTER SUMMARY

This chapter has given an overview of the existing solutions deployed by current DRM systems to sell licenses to consumers. It has also presented a detailed survey on the existing license reselling solutions proposed in the literature. It has highlighted their strengths and limitations. From this survey, we have identified a number of issues which should be addressed in a license reselling facility, i.e. (1) to support fair reselling, (2) to preserve current content owners’ rights, and (3) to accommodate buyers’ and resellers’ requirements. Finally, the chapter outlined our prospective ideas to design a fair and secure license reselling solution of DRM-protected contents.

To support fair reselling, the next chapter will introduce a contract signing protocol which is called Reselling Deal Signing (RDS) protocol.
Chapter 4

A Reselling Deal Signing (RDS) Protocol

4.1 Chapter Introduction

This chapter introduces a novel fair and abuse-free contract signing protocol known as Reselling Deal Signing (RDS) protocol. The RDS protocol is to support fair reselling of a DRM license without using a dedicated TTP. This protocol makes use of the concurrent signature (CS) scheme [8] and the existing license distribution infrastructure. By making use of the CS scheme and integrating it into the existing license distribution infrastructure, we avoid the use of a dedicated TTP, thus introducing no additional communication overhead while providing fair license reselling. A protocol is said to be fair if at the end of a protocol execution, either both involved entities obtain each other’s items or none of them gets anything useful. Also, the protocol is designed such that none of the two signers can prove to an outside entity that he is in control of the outcome of the protocol execution, thus achieving abuse-freeness. The abuse-freeness property [9] is required in a digital license reselling case to prevent Bob (the second signer, i.e. the buyer) from using the RD that is only signed by Alice (the first signer, i.e. the reseller) to gain any advantage by showing this RD to another reseller.

The chapter starts by giving a survey on existing approaches of contract signing protocols in Section 4.2. Section 4.3 presents a novel idea for the design of the RDS protocol. Section 4.4 gives the protocol design preliminaries (requirements, notations, and assumptions). In Section 4.5 and Section A.1 an overview and a description of the RDS protocol are given, respectively. In Section 4.6....
an informal analysis of the protocol is presented, while in Section 4.7 a formal verification of the protocol is given. This section first gives an overview of the formal methods in Section 4.7.1. It then, in Section 4.7.2 presents an overview of the Mocha model checker which has been used to verify the RDS protocol. The model of the RDS protocol and the verification results are then discussed in Section 4.7.3.2 and Section 4.7.3.3 respectively. In Section 4.8 we analyse the performance of the RDS protocol by (1) computing its computational cost, (2) comparing it with related work, (3) prototyping it and measuring its execution time. Finally, the chapter summary is given in Section 4.9.

4.2 A Survey of Fair Exchange Protocols

Contract signing protocols are a type of fair exchange protocols. This section gives a survey of existing fair exchange protocols. A fair exchange protocol should possess two important properties, fairness and non-repudiation. The fairness property means that at the end of an exchange, either both involved entities obtain each other’s exchanged items or none of them get anything useful. Non-repudiation is a security property that provides exchanged entities with undeniable evidence that a given item has indeed been sent (i.e. Non-repudiation Of Origin or NOO) or received (Non-repudiation of Receipt or NOR). An exchange protocol is said to be fair, if both non-repudiation and fairness properties are achieved.

Fair exchange protocols published in the literature can largely be classified into two categories. The first category is known as protocols without TTP, as they do not make use of any TTP to achieve fair exchange between the participants. The second category is referred to as TTP-based protocols, as they rely on a dedicated TTP to guarantee fairness.

4.2.1 Protocols without TTP

With the protocols without TTP, two entities exchange their respective items without making use of a dedicated TTP to achieve fairness. These protocols can be further classified into gradual exchange protocols and probabilistic protocols.
4.2. A SURVEY OF FAIR EXCHANGE PROTOCOLS

4.2.1.1 Gradual Secret Release Protocols

In the early 1980s, gradual secret release protocols [67, 68, 69, 70] were among the first fair exchange protocols proposed. These protocols typically work as follows. Two entities first engage in dividing their signatures into N parts, each of which being a verifiable component. They then exchange their respective signatures part-by-part. This exchange process continues until both entities have fully released their respective signature. To ensure fairness, it is assumed that the entities involved have equal computational power.

The contract signing protocol [70], proposed by Okamoto and Ohta, has used the gradual exchange approach. In this protocol, two entities, Alice and Bob, first agree on a contract, Cont. Each of Alice and Bob choose a secret, $S_A$ and $S_B$ respectively. Each of them then declares that they are committed to this agreed contract if each entity can know the secret of the other entity. In other words, Alice first signs a message saying that “I am committed to a contract Cont if Bob can show my secret $S_A$”. Bob acts in the same way. Alice and Bob then engage in the execution of a gradual exchange protocol to gradually (part-by-part) release their respective secrets $S_A$ and $S_B$.

The main advantage of this approach is that there is no involvement of a dedicated TTP during the protocol run. However, this approach is not practical for the following two reasons. Firstly, it assumes that both involved entities should have equivalent computational power. If one of them has a superior computing capability, then this approach can achieve neither fairness nor abuse-freeness. This is because the entity with a superior computing capability may terminate the protocol execution prematurely and use his resources to compute the remainder of the other entity’s secret. As a result, the other entity could be left in a disadvantageous position. Secondly, this approach requires: (1) N messages (each contains one part of the secret), and (2) a high level computational cost to compute and to verify each of the exchanged parts. Thus, it is inefficient in terms of communication overheads and computational cost.

4.2.1.2 Probabilistic Protocols

Another type of protocols that do not use TTP is probabilistic protocols [71, 72]. A probabilistic protocol is the one that can achieve probabilistic fairness. A probabilistic fairness means that an execution of a protocol can achieve fairness with a probability of $(1-\epsilon)$, where $\epsilon$ is set as a negligible value. To achieve an
adequate security level, however, an execution of a probabilistic protocol consist of a large number of repetitive rounds. Guilin [73] reported that the Markowitch-Roggenman protocol [71] (one of the probabilistic protocols) has to be repeated $2n + 2$ rounds of transmissions to achieve a security level of $\epsilon = 1/n$. This large number of rounds of transmissions makes this type of protocols inefficient as they need a high level of communication and computation costs.

We here use the probabilistic contract signing protocol proposed by Ben-Or et al. [71] to further explain the idea used by of this class of fair protocols. With the Ben-Or protocol, entities exchange privileges rather than bit information. It is said that an entity, Alice, is more privileged than the other, Bob, when Alice has a greater ability (than Bob) to convince an adjudicator\(^1\) that a contract has been signed by Alice and Bob. During the execution of a contract signing protocol, each entity is privileged in turn. Each entity sends to the other a message stating that with a probability, $\lambda$, a contract will be considered signed by both entities at a deadline, $T$. The probability, $\lambda$, must increase with each protocol round. The protocol execution ends when $\lambda = 1$ or when the deadline $T$ is expired. After the deadline $T$, each entity can present to the adjudicator the last received message. Upon the receipt of this message, the adjudicator selects a random value between 0 and 1 and matches this value with the probability, $\lambda$, extracted from the message. The adjudicator declares that the two entities committed to the contract if $\lambda$ is equal to or greater than the selected value. Otherwise, the adjudicator states that no entity is committed to the contract as $\lambda$ is too small.

Unlike the gradual secret release approach, the probabilistic approach does not impose the impractical requirement that the involved entities should have equal or similar computational power to achieve fairness. However, it requires a high number of rounds transmission protocol rounds to provide fairness. Thus, with this approach, communication and computational overheads are still high.

### 4.2.2 TTP-based Protocols

To overcome the problems associated with the protocols without TTP, TTP-based protocols are proposed [74, 75]. The main idea behind this class of protocol is that a dedicated TTP is invoked to provide fairness for an exchanging process. Depending on the level of the TTP’s involvement in the exchange process, the

\(^1\)An adjudicator is a natural judge who is only involved to resolve a dispute if the protocol is terminated prematurely. In this case, the adjudicator examines the signed messages
protocols of this category is further classified to three sub-classes, \textit{in-line TTP-based, on-line-based TTP,} and \textit{off-line TTP-based} protocols.

### 4.2.2.1 In-line TTP-based Protocols

With in-line TTP-based protocols \cite{76, 77, 78}, a TTP acts as an intermediary or a delivery authority in an exchange process. The TTP gathers exchanged items from the two involved entities. It then checks the correctness of these items and subsequently forwards them to the corresponding recipients.

\begin{center}
\begin{tabular}{c|c|c|c|c|c|c|c}
\hline
A & TTP & : & Msg\textsubscript{1} & = & M, & EOO  \\
A & TTP & : & Msg\textsubscript{2} & = & L, & EOS  \\
TTP & B & : & Msg\textsubscript{3} & = & L, & EOO  \\
B & TTP & : & Msg\textsubscript{4} & = & L, & EOR  \\
B & TTP & : & Msg\textsubscript{5} & = & L, & M  \\
A & TTP & : & Msg\textsubscript{6} & = & L, & EOR, EOD  \\
\hline
\end{tabular}
\end{center}

**Protocol Notations:**
- \( M \): message to be delivered,
- \( L \): unique label chosen by the TTP to identify message \( M \),
- \( \text{EOO} = \text{Sig}_A(M) \): evidence of origin,
- \( \text{EOS} = \text{Sig}_{\text{TTP}}(L, \text{EOO}) \): evidence of submission of \( M \) to TTP,
- \( \text{EOR} = \text{Sig}_B(L, \text{EOO}) \): evidence of reception of message labelled \( L \),
- \( \text{EOD} = \text{Sig}_{\text{TTP}}(L, \text{EOR}) \): evidence of delivery of \( M \).

Figure 4.1: Zhou et al protocol with in-line TTP

We can use the protocol proposed by Zhou et al. \cite{76} to illustrate the idea of using an in-line TTP. An overview of the protocol is given in Figure 4.1. An entity, Alice (A), initiates a protocol run by sending \( M \) and its \textit{EOO} in a message, \( \text{Msg}_1 \), to TTP. Upon the receipt of \( \text{Msg}_1 \), TTP verifies Alice’s signature in \textit{EOO}. If this verification is positive, TTP generates \textit{EOS} of \( \text{Msg}_1 \) and a unique label, \( L \), for \( \text{Msg}_1 \). TTP then makes \textit{EOS} and \( L \) public for Alice to fetch in \( \text{Msg}_2 \).

In \( \text{Msg}_3 \), TTP informs the second entity, Bob (B), that \( \text{Msg}_1 \) labelled with \( L \) received from Alice is ready for collection. If Bob decides to collect \( \text{Msg}_1 \), he sends \textit{EOR} to TTP in \( \text{Msg}_4 \). \textit{EOR} serves as Bob’s commitment to collect \( \text{Msg}_1 \). Once TTP receives \( L \) and \textit{EOR}, TTP generates \textit{EOD} of \( \text{Msg}_1 \) and publicizes all of \( \text{Msg}_1 \), \textit{EOD}, \textit{EOR} in a read-only directory. Bob can then use the label, \( L \) to fetch the message, \( M \) from the TTP’s directory, while Alice downloads \textit{EOD} in the same way.

With \textit{EOO}, Bob can verify that \( \text{Msg}_1 \) has been generated by Alice. With \textit{EOS}, Alice can prove that TTP has indeed received \( \text{Msg}_1 \), whereas \textit{EOR} proves
that Bob has committed to collect $Msg_1$, and $EOD$ proves that $Msg_1$ has been indeed publicized for Bob to retrieve.

Although the in-line TTP-based protocols are simple, they suffer from several disadvantages. Firstly, they require TTP to maintain and manage large databases of data and exchanged evidence. TTP must preserve all messages it receives. Managing these databases places a significant level of responsibility including security on TTP. Secondly, as TTP is involved in every step of the execution of the exchange protocol and its availability is very important for the success of the protocol, it is prone to become a communication and computation bottleneck. In addition, TTP could be at risk of becoming a security bottleneck. This is because a compromise of TTP would make all the transactional evidence and the exchanged items at risk. Thirdly, TTP can easily get access to the items being exchanged and any transactional evidence. This puts the confidentiality of the exchanged items at risk from. This is why in this class of protocols, TTP has to be an unconditionally trustworthy entity.

4.2.2.2 On-line TTP-based Protocols

To overcome the weakness of in-line TTP-based protocols, on-line TTP-based protocols [4, 79, 80, 81] have been proposed. With this latter approach, TTP does not act as a delivery authority, rather it plays an assistant role to ensure fairness. In other words, an on-line TTP does not involve in every message transmission. It generates, validates, and/or stores evidence of transactions for the involved entities. The transitioned message is encrypted by a symmetric key and sent directly from a sender to a receiver.

To further explain the idea of the on-line TTP-based approaches, an overview of the Zhou and Gollmanns protocol [4] is given below. In this protocol, Alice starts a protocol execution by first ciphering her message, $Msg_1$, using a session key $k$ shared between Alice and Bob. Alice then sends Bob this ciphertext directly along with $EOO$ for the ciphered message to a recipient, Bob. Bob then confirms the reception of the ciphertext by sending $EOR$ of the ciphered message to Alice. Upon the receipt of $EOR$, Alice signs the decryption key, $Sub_k$, and lodges it with the TTP. After receiving this key and $Sub_k$, the TTP notarises the key submission by Alice by generating a confirmation of the key, $Con_k$.

Bob will then fetch $k$ and $Con_k$ from the TTP, and Alice will receive $Con_k$ from the TTP as well. With this kind of protocols, a non-repudiation of origin
4.2. A SURVEY OF FAIR EXCHANGE PROTOCOLS

Figure 4.2: Zhou and Gollmann’s protocol [4] with on-line TTP

Evidence consists of two items: EOO and Conk. Also, a non-repudiation of receipt evidence consists of two items: EOR and Conk.

It can be seen that the TTP is not involved in the first two messages of the protocol run. In other words, the TTP does not act as a deliver authority as in the case of the in-line TTP-based approach, but only provides the decryption key to the participants.

With the introduce of the on-line TTP-based protocols, the level of TTP’s involvement is minimised in comparison with the in-line TTP-based protocols. However, with the on-line TTP-based protocols, the on-line TTP is still required to be involved in every protocol run. In addition, it is required to host a secure public server for the involved entities to post/fetch their messages or decryption keys. These operations make these protocols subject to Denial of Service (DoS) attacks. This means that the TTP of this solution can still be a bottleneck problem in terms of performance and security.

More efforts have been made to further minimise the level of TTP’s involvement. These efforts have resulted in a third approach known as off-line TTP-based approach.

4.2.2.3 Off-line TTP-based Protocols

Off-line TTP-based protocols [82, 83, 84, 85, 86, 87, 88, 89, 90] aim to further reduce the involvement of TTP. A TTP is said to be off-line if it only intervenes
in an exchange protocol if a problem takes place. A problem could occur if a protocol participant misbehaves or there is a network error. If such a problem happens, TTP is invoked to help the participants to finish the protocol run in a fair way. With off-line-based protocols, it is supposed that most of the time, a protocol is successfully executed without any problems. For this reason, protocols with off-line TTP are also known as optimistic protocols.

An off-line TTP-based protocol typically consists of two sub-protocols: a main exchange protocol and a recovery protocol. With the main exchange sub-protocol, involved entities attempt to exchange their items without invoking the TTP. If they fail to do so, one of them invokes the TTP by launching the recovery sub-protocol. With this sub-protocol, the TTP resolves this fail by recovering the disputed items and enforcing fairness.

Some off-line TTP-based protocols, e.g. the one described in [85], have an extra sub-protocol, called an abort protocol. An abort sub-protocol is executed with a TTP if an entity wants to terminate the execution of a main exchange sub-protocol before its normal end. With this sub-protocol, TTP digitally signs an affidavit to confirm that the protocol has been aborted.

An off-line TTP-based protocol may support one of two types of fairness, weak fairness and strong fairness. A weak fairness means that the fairness can be only achieved, if the TTP can generate, during the recovery sub-protocol, an affidavit confirming what happened during the main exchange. The disadvantaged entity will be given this affidavit to use in an external dispute resolution system, e.g. a court of law, to prove that the other entity has misbehaved and reinforce fairness. A strong fairness can be achieved when either both involved entities have received what they expect, or neither entity receives anything useful. In other words, if, during the protocol run, a TTP can revoke an exchanged item, which has already been released during the main exchange, or can generate a replacement for an exchanged item, which failed to arrive to intended recipient during main exchange, then a protocol with off-line TTP can guarantee strong fairness.

To illustrate the idea of the off-line TTP-based approach, an overview of the Asokan and Shoup’s contract signing [85] is given below. As depicted in Figure 4.3, the protocol consist of three sub-protocols: exchange, abort and recovery. With the exchange sub-protocol, two parties, Alice (A) and Bob (B), exchange their respective signatures. Alice first generates her signature on both contract Cont and her nonce $N_A$ to form a token denoted as $me_1$. She then sends $me_1$ to
4.2. A SURVEY OF FAIR EXCHANGE PROTOCOLS

Exchange sub-protocol:

\[
\begin{align*}
A &\rightarrow B : \text{Msg1:=} \text{SigA}(C,h(NA)) \\
B &\rightarrow A : \text{Msg2:=} \text{SigB(me1,h(NB))} \\
A &\rightarrow B : \text{Msg3= NA or A invokes a bort sub-protocol (if me2 fails to arrive)} \\
B &\rightarrow A : \text{Msg4= NB or B invokes recovery sub-protocol (if NB fails to arrive)} \\
\text{If A timeouts waiting for NB, then A invokes recovery sub-protocol} \\
\text{Final contract is of the form (Msg1,NA,Msg2,NB).}
\end{align*}
\]

A bort sub-protocol:

1. A \rightarrow TTP : \text{SigA(aborted, Msg1)}
   
   If recovered then ma := \text{SigTTP(Msg1,Msg2)}
   
   else aborted :=true; ma := \text{SigTTP(aborted,SigA(aborted,Msg1))}

2. TTP \rightarrow A,B : ma

Recovery sub-protocol:

1. A or B \rightarrow TTP : \text{Msg1, Mdg2}
   
   If aborted then mr := \text{SigTTP(aborted,SigA(aborted,Msg1))}
   
   else recovered :=true; mr := \text{SigTTP(Msg1,Msg2)}

2. TTP \rightarrow A,B : mr

\text{Final contract is of the form } \text{SigTTP(Msg1,Msg2).}

Protocol Notations:

- C: contract to be signed.
- NA: nonce generated by Pa.
- NB: nonce generated by Pb.
- SigA(aborted,Msg1): a signed abort request.
- ma := \text{SigTTP(Msg1,Msg2)} or ma := \text{SigTTP(aborted,SigA(aborted,Msg1))}.
- mr := \text{SigTTP(Msg1,Msg2)} or mr := \text{SigTTP(aborted,SigA(aborted,Msg1))}.
- \text{valid final contract is either (Msg1,NA,Msg2,NB) or SigTTP(Msg1,Msg2)}

Figure 4.3: Sokan and Shoup’s contract signing protocol with an off-line TTP

Bob. Bob then produces his signature on both \text{ma}_1 and his nonce \text{N}_B to from \text{me}_2. Bob then, in \text{Msg}_2, send \text{me}_2 to Alice. In \text{Msg}_3, Alice will reply with \text{N}_A to Bob. Optimistically, Bob will also send \text{N}_B to Alice in \text{Msg}_4. At this point, both Alice and Bob have received a jointly signed contract of the form \text{[me}_1,\text{N}_A,\text{me}_2,\text{N}_B].

An \text{abort} sub-protocol is invoked if Alice has sent \text{Msg}_1 but did not receive \text{Msg}_2 from Bob. With this protocol, upon receiving an abort request (\text{Req}_a) from Alice, a TTP will sign and sends both entities an abort-token, \text{ma}, to revoke \text{me}_1 which has already been released.

A \text{recovery} sub-protocol is invoked if either \text{N}_A or \text{N}_B has failed to reach its intended recipient. For example, if Bob does not receive \text{N}_A for any reason, he should invoke the recovery sub-protocol with TTP by sending (\text{me}_1 and \text{me}_2) to
the TTP. If the sub-protocol has not already been aborted, TTP generates its signature in \((me_1 \text{ and } me_2)\) and sends it to both of Alice and Bob as a replacement of the final signed contract. Alice can do the same procedure if she does not receive \(N_B\).

Recently, a sub-class of the off-line TTP-based protocols has been introduced, which is called a transparent TTP \([82, 83, 90]\). With this type of protocols, at the end of a protocol execution, it is impossible to tell whether TTP has been indeed involved in the protocol execution. To achieve this property, the protocol makes use of a special cryptographic primitive called Verifiable and Recoverable Encrypted Signatures (VRES). The VRES represents a digital signature which is encrypted in such a way that its receiver can verify:

- the correctness of the encrypted signature without having any information about the original signature, i.e. verifiability, and
- in the case where the original signature sender refuses to send his original signature, the designated TTP can help to recover it from the VRES (i.e. recoverability).

Although the use of the TTP-based approach is simple, and it has been made efficient recently, TTP could results in one of the following problems:

- The TTP may become a performance and security bottleneck leading to Denial of Service (DoS) attack.
- When the TTP is involved in a protocol run, it decreases its efficiency.
- As TTPs provide services for the participants, they need to be paid for such services \([15]\). Consequently, the transaction cost will be increased.
- Last but not least, it may be difficult to find a third party which is trustworthy and could serve as TTP all the time.

To address these TTP-associated problems, a new signature scheme called Concurrent Signature \([8]\) is suggested.

### 4.2.3 Concurrent Signature (CS) based Protocols

This class of signature fair exchange protocols makes use of The CS scheme presented in \([8]\) and \([91]\). This scheme, as explained in Appendix B, allows two
entities to generate and exchange two ambiguous signatures, which are not binding to their respective signers until the release of a keystone, $ks$, (an extra piece of information). Upon the release of the keystone, both signatures become binding to their respective signers concurrently, thus achieving fair signature exchange. Other variants of the CS scheme can also be seen in the literature, e.g. an identity-based concurrent signature \[92\] and a concurrent signature with negotiable binding control \[93\].

The CS scheme is very practical in the sense that, to fairly exchange signatures, (1) it does not require the signers to have the same level of computational power as in the case of gradual exchange approach; (2) it does not need many rounds of messages transmission as in the case of gradual secret release and probabilistic fairness approaches; (3) it does not require any assistance of a TTP.

However, as reported in \[8\], the CS scheme can only provide a weak fairness. With the CS scheme, only one of the two entities could decide whether the signatures exchange process can be fairly completed. The entity holding the keystone can decide when and whether the keystone can be released (i.e. sent to the other entity). Thus, if the keystone is not released, neither fairness nor abuse-freeness can be achieved.

4.3 A Novel Idea for The RDS Protocol

Neither protocols without TTP nor TTP-based protocols are readily applicable to a signing of an RD contract in a digital license reselling process. The gradual secret release protocols, the first type of the protocols without TTP, are not suitable due to the following two reasons. The first reason is that it is not practical to assume that both a reseller and a buyer have equal computational power. In real life, the reseller could use a desktop computer while the buyer may use a mobile phone, in which case neither fairness nor abuse-freeness can be achieved. The second reason is that the gradual secret release approach imposes a high level of communication and computation costs as it requires: (1) N message flows to exchange the N parts of the entities’ signatures, and (2) the computation and verification of each of the exchanged parts. Thus, it is inefficient in terms of communication overheads and computational cost. For similar reasons, the probabilistic protocols (the second type of the protocols without TTP) are also not suitable to sign an RD contract in a digital license reselling.
Also, TTP-based protocols cannot be adopted to sign a RD contract with a license reselling process because of the following reasons. In a DRM infrastructure, there is already a trustworthy entity, i.e. LI, to support a license reselling process \[22\]. Introducing another TTP in order to support fairness in this infrastructure would add more cost into a transaction. One could say that LI itself could be used to serve as a TTP in the signing process to help to achieve fairness. Indeed, this should be planned out carefully. Otherwise, the following scenario may occur. For example, if the TTP-based protocol proposed by Nenadic et al \[94\], (the authors report that this protocol is efficient in comparison with related work), is used, and LI serves as the TTP during the signing process, LI will need to perform the following additional tasks: (1) prior to executing a signing process, LI has to issue and sign a special digital certificate to certify an additional public/private key pair for the initial signer; (2) in case of a dispute, LI has to perform a number of signature verifications in order to resolve the dispute \[94\]. These tasks would add on to the existing workload which LI is already performing in the current license distribution processes.

In addition, if Nenadic’s protocol is used to sign a given RD, the communication overhead of the reselling process would also be increased. This is because: (a) Nenadic’s protocol requires 7 messages: 4 messages for main exchange protocol and 3 for recovery protocol; (b) further messages (at least 2 messages) are required for the initiator to get a digital certificate from LI before he initiates a protocol execution with the other party. Thus, if the RD contract signing process is performed normally between a reseller and a buyer, LI must engage in the 2-message protocol to issue the reseller (the initiator) the special digital certificate. If there is any dispute, LI will need to engage in executing the recovery protocol consisted of 3 further messages. This means that LI, in the worst case scenario, will need to send and receive 5 messages during the process of signing an RD. This would increase the communication overhead of a license reselling process.

In their work \[86\], Zhang et al have proposed an optimistic exchange protocol based on bilinear pairing. They have also proven that their protocol is more efficient than Nenadic’s one \[94\]. However, if Zhang’s protocol is used to sign a given RD in a reselling process, the communication overhead would be increased as well. This is because Zhang’s protocol requires three sub-protocols to achieve fairness. It requires 4 messages for the main exchange protocol, and additional 3 messages for both the abortion and dispute resolution. In other words, if this
4.3. A NOVEL IDEA FOR THE RDS PROTOCOL

A protocol is used in our case [22], LI may have to execute a protocol of 3 messages either for the abortion operation or for the dispute resolution.

The communication and processing overheads imposed on LI can be reduced if the RD signing protocol makes use of the Concurrent Signature (CS) scheme [8]. However, as reported in [8], the CS scheme, described in Appendix B, can only provide a weak fairness. Recalling our discussion of the fairness property can be assured under two conditions: (1) the initial signer releases a secret token, i.e. a keystone, $k_s$, at the end of the exchange, and (2) the initial signer does not abuse a pre-binding token signed by the other signer before the completion of the exchange process (i.e. it can provide the abuse-freeness property).

We also observed that in the existing license distribution infrastructure, the distribution of licenses is coordinated via LI. In other words, we remarked that the services and properties offered by LI and CS are complementary. If we integrate the CS scheme with the existing license distribution infrastructure (i.e. LI), which is already used to issue digital licenses, we can then get the best out of the two components LI and the CS scheme. The idea is that we can design a contract signing protocol by integrating the CS scheme with the functions already offered by LI. This contract signing protocol can provide the fairness and the abuse-freeness properties. In addition, it does not require a dedicated TTP, and the overhead introduced would be less than that required by other approaches. Based on this motivation, we have designed a contract (Reselling Deal) Signing (RDS) protocol. This protocol can overcome the two weaknesses of the CS scheme, described above by integrating LI functionalities with the CS scheme. The role of LI in the RD signing process is the same as the role played by LI in the license reselling process described in chapter 5. In other words, LI is only involved in one case, which takes place if an RD has been signed, and Bob (the buyer) wants to activate the license stated in the signed RD. In this case, which is called RD activation process, LI is invoked to receive a payment from Bob and to verify both entities’ signatures on the RD. LI then sends Bob the license and sends Alice the payment. This means that LI is only involved in the RDS protocol if a license will indeed be resold. By this way, LI can ensure fairness and abuse-freeness to prevent cheating by either Alice or Bob in the middle of a signing process.

The RDS protocol can also be used to enable a consumer, $C_1$, who is subscribed to access a media website, to fairly resell her access permissions to another consumer, $C_2$. In such a scenario, $C_1$ can use the RDS protocol to sign a deal.
with $C_2$. This signed deal can then be used by $C_2$ to claim $C_1$’s access permissions from the owner of the website. This can only be done should $C_2$ make a payment stated in the signed deal to the owner. Also, $C_1$ can use this deal to claim the agreed payment paid to the owner by $C_2$.

The remaining part of this chapter is dedicated to the description, informal and formal analysis, end valuation of the RDS protocol.

4.4 Preliminaries

4.4.1 Notations

The notations used in the RDS protocol are given below.

1. $A$, $B$, and $LI$: Alice, Bob and License Issuer.

2. $Lic$: A resalable license that Alice has bought from LI and she wants to resell to Bob.

3. $RD$: A contract, called Reselling Deal (RD), that has been agreed by Alice and Bob.

4. $ks$ and $f$: Keystone and its keystone fix, respectively. They are used in the signing process.

5. $RP_{Lic}$: A reselling permission issued for $Lic$. It proves that $Lic$ is resalable. It is of the form $RP_{Lic} = \{Lic||f||Sig_{LI}(Lic||f)\}$.

6. $PK_i$ and $SK_i$: Public and private keys of entity $i$.


8. $E_{PK_i}$: Asymmetric encryption using entity $i$’s public key.

9. $ASig_i$: An ambiguous signature created by entity $i$.

10. $Sig_i$: A digital signature created by entity $i$.

4.4.2 Design Assumptions

The RDS protocol has used the following assumptions.

1. Similar to the current DRM systems, LI is assumed to be a trustworthy entity which is responsible for issuing a license and facilitating a license reselling process.

2. Alice has got a license, Lic, with a reselling permission, $RP_{Lic}$, and a keystone, $ks$, from LI. We assume that $RP_{Lic}$ and $ks$ for a given license are always embedded in the original license. Thus, when Alice bought Lic from LI, she should have been issued with $ks$ and $RP_{Lic}$ that contains the keystone fix, $f$. This means that there is no need for LI to perform any additional operation with Alice before she starts an execution of the RDS protocol.

3. The buyer can make payment for the license to LI in a secure manner and the reseller can receive the payment from LI in a similarly secure manner.

4. Keystone, $ks$, can only be used once.

5. Hash functions are collision-free.

6. Alice and Bob have agreed on the terms and conditions of a reselling deal, RD.

7. Each entity, $E_i$, where $(i \in A, B, LI)$, has a public/private key pair, $PK_i/SK_i$. $PK_i$ has been certified by a certification authority and $SK_i$ is kept secret by their respective holders.

8. Communication channels between any pair of entities are resilient.

9. Communication channels are authenticated, confidential and integrity-protected. These channels can be established using the Secure Socket Layer (SSL) protocol [95].

4.4.3 Design Requirements

The RDS protocol is designed to satisfy the following requirements.
• (R1) Non-repudiation of signature origin - the recipient of a signature is assured that the signature is indeed generated by the claimed signer.

• (R2) Non-repudiation of signature receipt - the sender of a signature is provided with evidence that the intended recipient has indeed received this signature.

• (R3) Abuse-freeness: before the completion of a protocol execution, neither of the signers can prove to an outside entity that he is in control of the outcome of the protocol execution.

• (R4) Strong fairness: upon the completion of a protocol execution or upon an acceptance of an RD activation request, either both the signers are committed to the contract or neither is.

4.5 RDS Protocol Overview

As shown in Figure 4.4, the protocol consists of three messages: \( Msg_1 \), \( Msg_2 \), and \( Msg_3 \). In \( Msg_1 \) and \( Msg_2 \), Alice and Bob exchange their respective ambiguous signatures on a given RD. Alice first creates \( \text{ASign}_A \), where \( \text{ASign}_A = \text{ASign}_A(RD) \), and sends it along with \( RP_{\text{Lic}} \) to Bob. Upon receiving \( Msg_1 \), Bob checks the license ID by performing \( B_{V1} \). Bob then verifies the correctness of \( RP_{\text{Lic}} \) and \( \text{ASign}_A \) by performing \( B_{V2} \) and \( B_{V3} \), respectively. If these verifications are positive, Bob creates \( \text{ASign}_B \), where \( \text{ASign}_B = \text{ASign}_B(RD||\text{ASign}_A) \). The purpose for Bob to sign \( RD_{\text{ASign}_A} \), instead of signing on the RD directly, is to prevent Alice from abusing \( \text{ASign}_B \) once she receives it in \( Msg_2 \) (i.e. to achieve abuse-freeness). He then sends it to Alice. After receiving \( \text{ASign}_B \), Alice verifies its correctness using \( A_{V1} \) and \( A_{V2} \) and then, in \( Msg_3 \), sends \( ks \) to Bob. Upon releasing \( ks \), Alice obtains Bob’s binding signature, i.e. \( (ks, \text{ASign}_B) \) and Bob gets Alice’s binding signature, i.e. \( (ks, \text{ASign}_A) \), thus achieving fair RD signing process.

Upon the successful completion of a protocol execution, both Alice and Bob (1) can not falsely deny having created their respective signatures, \( (ks, \text{ASign}_A) \) and \( (ks, \text{ASign}_B) \), thus achieving non-repudiation of signature origin, (2) will have a proof that each one of them has received the other’s signature (Bob can use \( ks \) as a proof that Alice has indeed received \( \text{ASign}_B \) sent in \( Msg_2 \) and Alice \( \text{ASign}_B \) as a proof that Bob has certainly received \( \text{ASign}_A \) in \( Msg_1 \)), achieving
4.6 RDS Protocol Informal Analysis

In this section, the RDS protocol is analysed against the requirements set in Section 4.4.3. In a license reselling process, after RD has been signed by a reseller and a buyer, LI has to be invoked by the buyer to activate this signed RD. This activation process can deter both the reseller and the buyer from misbehaving during the RD sign process. As indicated below, it is hard for any entity to benefit from misbehaving. During the activation process, LI will perform the following verifications and will only accept a given RD if they are all positive.

1. \( (LI_{V1}) \): Confirming that the payment, \( P \), equal to the amount stated in RD, has been made by Bob and verifying Bob’s signature on the RD activation request.

2. \( (LI_{V2}) \): Checking that \( Lic \) is legitimate to resell. This check consists of two further checks: \( LI_{V2,1} \) and \( LI_{V2,2} \). In \( LI_{V2,1} \), LI verifies whether \( Lic \) has a valid RP. This is done by verifying \( \text{Sig}_{LI}(Lic||f) \). If this signature is invalid, it means that either \( Lic \) or \( f \) is incorrect. Hence, \( Lic \) is deemed as non-resalable and LI will reject RD. If the signature is valid, \( Lic \) is considered as resalable. LI then proceeds to perform \( LI_{V2,2} \) in which LI ascertains whether \( ks \), corresponding to \( f \), has already been released. If \( ks \) has already been publicised, it means that \( Lic \) has already been resold. So, LI will reject RD. Otherwise, LI proceeds to perform \( (LI_{V3}) \).

3. \( (LI_{V3}) \): Checking whether the license identity, \( Lic \), specified in \( RP_{Lic} \), is identical to the license identity, \( Lic \), specified in RD. This check can prevent Bob from replacing \( RP_{Lic} \) with another less valuable reselling permission, \( RP_{Lic2} \).

4. \( (LI_{V4}) \): Verifying \( A\text{Sign}_{B} \) to confirm Bob has signed RD.

5. \( (LI_{V5}) \): Verifying \( A\text{Sign}_{A} \) to ensure Alice has signed RD.
4.6.1 Fairness Analysis

This section analyses the fairness property of the RDS protocol. Figure 4.5 shows every possible action Alice or Bob could take when she or he, respectively, creates and/or receives a protocol message. This section introduces analysis of all these possible actions. The analysis assumes that any payment, $P$, made to LI is equal to the amount stated in the submitted RD. Otherwise, this RD will be rejected.

The discussions are structured based on three cases (a) Alice and Bob behave properly; (b) Alice misbehaves; and (c) Bob misbehaves.

**Alice and Bob behave properly**: As depicted in Figure 4.5, there are two scenarios for Alice and Bob to achieve fairness. These two scenarios are depicted as dashed lines in the figure. In the first scenario, Alice and Bob honestly follow the protocol described in Section A.1.

In the second scenario, like in the first scenario, Alice sends $Msg_1$ to Bob who may be very happy with the RD to be signed. Thus, instead of sending $Msg_2$ to Alice, Bob may construct an RD activation request and sends it to LI.

---

2 In the context of the RDS protocol, fairness means that at the end of a deal signing process undertaken between a reseller and a buyer, either the reseller receives a deal signed by the buyer and the buyer receives a deal signed by the reseller, or neither of them receives anything useful.
This request is of the form \( \{Msg_1|Msg_2||P\} \) or \( \{Msg_2||RP_{Lic}||P\} \). Although the signatures in \( Msg_1 \) and \( Msg_2 \) are still ambiguous, LI can use \( ks \) to convert them into binding signatures. LI can then activate the RD. In this activation, LI sends Bob the activated license \( Lic \), makes \( ks \) public and sends Alice the payment, \( P \). In this case, Alice will not be affected as her license has been resold (this is the main aim of designing the RDS protocol). She can also get the signed RD along with the payment. Thus, in the second scenario, both Bob and Alice can obtain a signed RD (i.e. achieving fairness).

**Alice misbehaves:** As shown in Figure 4.5, Alice may misbehave in three occasions. In the first occasion, when Alice creates \( Msg_1 \), she may take two malicious actions against LI: A2 and A3. A malicious action here means to send a fake RD activation request to LI in attempt to cause DoS (Denial of Service) attacks against LI. For both of these actions, upon performing LI\(_{V_1}\), LI will reject the RD request as A2 and A3 do not contain a payment. In others words, we
have made the payment verification as the first one in the verification chain to minimise the risk of DoS attacks. In addition, Alice cannot gain anything useful by taking A2 and A3.

In the second occasion, upon the receipt of $Msg_2$, Alice may take one or more of five malicious actions (i.e. A7, A8, A9, A10, and A11). In all these actions, as shown in Figure 4.5, Alice may send some incomplete RD activation requests to LI. However, as Alice did not provide a payment in these actions, she will gain nothing by taking any of these actions. This is because, upon performing $LI_{V_1}$, LI will reject these RD submissions.

In the third occasion, after creating $Msg_3$, Alice may perform two misbehaving actions with Bob: i.e. A5 and A6. In A5, Alice, in $Msg_3$, may send Bob $ks_1 \neq ks$. If this happens, Bob will not be affected as he can detect this misbehaviour when he performs $B_{V_4}$. Bob then has three choices: (1) ask Alice to resend the correct $ks$; (2) terminate the protocol run as Bob may no longer be interested in completing the signing process of this RD with Alice (for example, he may have got another deal which is better than that offered by Alice); or (3) send an RD activation request to LI to activate the license and get the $ks$ with the activated license. In A6, Alice may abort the protocol run. However, this abortion will not affect Bob as he can still be able to activate the agreed RD with LI. He can send LI an RD activation request containing $\{Msg_1||Msg_2||P\}$, in which case, he can get $ks$ along with the activated license from LI. In any of the above cases, Bob will not experience any loss as he can still able to activate RD.

**Bob misbehaves**: As shown in Figure 4.5, Bob may misbehave in two occasions. In the first occasion, once Bob receives $Msg_1$, he may take one of two malicious actions: (a) either submit B5 or B6 to LI as RD activation requests, and (b) misbehave with Alice (i.e. send Alice B10). If Bob submits B5 or B6 to LI, LI will reject his submission as $LI_{V_1}$ and $LI_{V_4}$ will be negative, respectively. Therefore, if Bob does any of B5 or B6, he can neither bring any harm to Alice nor gain anything useful for himself. Bob can do B10 as it is his right to abort the protocol run if he does not wish to go for the deal. However, if this happens, similar to the case of B5 and B6, Alice will not be affected as, at this stage, Bob has only got $ASign_A$ which is not yet binding to Alice.

In the second occasion, upon the creation of $Msg_2$, Bob may take one of four malicious actions against LI, i.e submitting B4, B7, B8, or B9 to LI to activate the RD. In B4, although Bob has submitted an RD activation request containing
all the required elements, upon the verification of $LI_{V3}$, LI will reject this request. In $B7$ and $B8$, LI will also reject these requests as $LI_{V1}$ will produce a negative result. In $B9$, though the payment is included, after performing $LI_{V2,1}$, LI will reject Bob’s submission as $RP_{Lic}$ is not provided. Thus, by doing $B4$, $B7$, $B8$, or $B9$, Bob will not gain any benefit. In addition, since Bob has only got an ambiguous signature, $ASign_A$, from Alice, he can not bring any harm to Alice using this signature.

From the above discussions, three remarks can be drawn. Firstly, Alice can not bring any harm to Bob if she refuses to send $ks$ to Bob in $Msg_3$. This is because once Bob has received $Msg_1$ and created $Msg_2$, he can activate RD directly with LI and this activation can only be successful if Bob sends LI $\{Msg_1||Msg_2||P\}$ which include the payment. Secondly, Bob can not gain any advantage by first receiving $ASign_A$ as $ASign_A$ is not binding to Alice before Alice releases $ks$. Thirdly, all Alice’s submissions to LI will be rejected unless she provides a payment. Of course, it does not make any sense that Alice, as the reseller, would make any payment to resell her license.

4.6.2 Non-repudiation Analysis

This section analyses the RDS protocol against the non-repudiation requirements, i.e. the non-repudiation of origin (NOO) and non-repudiation of receipt (NOR) of signatures being exchanged.

In our problem context, NOO means that upon the successful completion of a protocol execution, both Alice and Bob can not falsely deny having created their respective signatures, $(ks, ASign_A)$ and $(ks, ASign_B)$. When $ks$ is released in $Msg_3$, both $ASign_A$ and $ASign_B$, exchanged in $Msg_1$ and $Msg_2$, will become binding to their respective signers. Hence, neither entity can falsely deny having created their signatures. If $ks$ is not released or if what is released is not identical to the keystone used in the signing process (i.e. if $ks_1 \neq ks$), then only Alice will have $(ks, ASign_B)$ and Bob will not have $(ks, ASign_A)$. In this case, Alice could deny having signed RD. However, if Bob submits $\{Msg_1||Msg_2||P\}$ to LI for activating RD, he will receive $ks$ with the activated RD from LI. Bob then uses this $ks$ to obtain $(ks, ASign_A)$. This means that once $ks$ is publicised, neither Alice nor Bob could falsely deny having created their respective signatures. Consequently, our protocol meets the NOO requirement.

Regarding NOR of the signatures, upon the successful completion of a protocol
CHAPTER 4. A RESELLING DEAL SIGNING (RDS) PROTOCOL

run, both Alice and Bob will have a proof that each one of them has received the other’s signature. For Bob, once he receives $ks$ in $Msg_3$, this $ks$ serves as a proof that Alice has indeed received $ASign_B$ sent in $Msg_2$. This is because Alice only releases $ks$ once she receives $ASign_B$. If Bob does not receive $ks$, he will not be able to tell that Alice has indeed received $ASign_B$. However, if Bob sends LI an RD activation request of the form $\{Msg_1||Msg_2||P\}$, Alice will receive $ASign_B$ from LI during the RD activation process. In other words, a successful RD activation process ensures Bob that Alice has indeed received $ASign_B$ when LI sends Alice both the payment and the signed RD.

For Alice, once she receives $ASign_B$ in $Msg_2$, she can use this $ASign_B$ as a proof that Bob has certainly received $ASign_A$ in $Msg_1$. This results from the idea that $ASign_A$ is a part of $ASign_B$. If Alice does not receive $ASign_B$, she will not be certain that Bob has indeed received $ASign_A$. However, when Bob sends LI an activation request of the form, $\{Msg_1||Msg_2||P\}$, Alice will receive $ASign_B$ from LI during the RD activation process. So, this request to LI serves as a proof that Bob has indeed received $ASign_A$.

From the above discussions, it can be concluded that upon a successful execution of the protocol or upon a successful RD activation, both Alice and Bob will have received the other party’s signatures. This implies that our protocol meets NOR requirement.

4.6.3 Abuse-freeness Analysis

The section shows that the RDS protocol is abuse-free. We discuss whether Alice or Bob could gain any benefit by showing any intermediate result of a RDS protocol run to a third party. In fact, the RDS protocol produces two intermediate results: (1) RD ambiguously signed by Alice, i.e. $RD_{ASign_A}$, and (2) RD ambiguously signed by both Alice and Bob, i.e. $[RD_{ASign_A}||ASign_B]$. $RD_{ASign_A}$ is received by Bob in $Msg_1$. $[RD_{ASign_A}||ASign_B]$ is received by Alice in $Msg_2$.

For $RD_{ASign_A}$, Bob can not abuse it to gain an advantage over Alice. Of course, Bob can show this result to another reseller, Eve. However, Bob can not prove to Eve that $RD_{ASign_A}$ is indeed signed by Alice as $ASign_A$ is an ambiguous signature and it could have been produced by Bob. Thus, Bob can not abuse $RD_{ASign_A}$.

With respect to $[RD_{ASign_A}||ASign_B]$, although it is an ambiguous signature, Alice is able to make it a binding signature to Bob and then show it to a third
4.6. RDS PROTOCOL INFORMAL ANALYSIS

party, Carol. As Alice holds the $ks$, she can use it to prove to Carol that $[RD_{ASign_A}]_{ASign_B}$ is indeed signed by Bob. However, Carol can also see that this RD has also been signed by Alice as $ASign_A$ is a part of $ASign_B$ (see Section A.1). Hence, it would be difficult for Alice to gain anything useful by doing so as it does not make sense for Alice to show Carol RD which is binding to both of Alice and Bob. So, we say that Alice can not abuse $[RD_{ASign_A}]_{ASign_B}$.

From this analysis, we can conclude that neither Alice nor Bob can abuse each others’ signatures if the protocol is not successfully completed. Thus, the RDS protocol satisfies the abuse-freeness property.

4.6.4 Security Analysis

As mentioned early in this Chapter, the CS scheme is the main cryptographic primitive used in the RDS protocol design. Hence, the security of the RDS protocol lies in the security of the CS scheme [8]. In other words, the security level of the RDS protocol is the same as the security level of the CS scheme provided that LI is trustworthy. In addition, since communications between entities A and B are carried out through a confidential channel (owing to assumption 8), the exchanged data is not exposed to any outsiders. This prevents any intruder from gaining anything useful from the messages while they transmit through the network. SSL protocol also protects against any replay attack [96].

Also, it is crucial for the security of the RDS protocol to ensure that the keystone $ks$ is not reused. If $ks$ is reused, Alice will be able to resell her license, Lic, multiple times. Consequently, the security of the DRM system will be compromised, i.e. the content owner’s rights will be violated.

The authenticity of the RD contract is protected by Alice’s and Bob’s signatures. As Alice signs RD first, if she has modified the negotiated RD before signing and sending it to Bob, Bob can detect such modification and then he can refuse to sign this RD. Also, as Bob is responsible for sending an RD activation request to LI, Bob would be able to modify RD before he sends the request to LI. However, as Bob has to jointly sign RD, which has been already signed by Alice, Bob can not modify the RD unless he can create Alice’s signature on this modified RD. In other words, in order for a given RD to be activated, this RD has to be jointly signed by both Alice and Bob and none of them would be able to make any unilateral alteration to RD without being detected.

The authenticity of $RP_{Lic}$ is protected by LI’s signature. If Bob attempts to
modify it, during performing $LI_{V2}$, $LI$ can detect such modification. $LI$ can then reject Bob’s RD activation request.

With our protocol, either Alice or Bob could replace a reselling permission, $RP_{Lic}$, with another less valuable reselling permission. Firstly, if Alice has committed this misbehaviour, with verification $B_{V1}$, Bob can detect that the license ID contained in the deal, RD, and in the permission, $RP_{Lic}$ are not identical. Bob can then stop the protocol execution. Secondly, if Bob has replaced $RP_{Lic}$ with another permission, $LI$ can detect this misbehaviour while performing verification $LI_{V3}$. $LI$ can then reject the RD activation process. Thus, neither Alice nor Bob can gain anything useful by doing this misbehaviour.

4.7 RDS Protocol Formal Verification

In this section, the RDS protocol is formally verified. Prior to the formal verification of the RDS protocol, we give a brief overview of formal verification methods that may be used in verifying fair exchange protocols.

A formal analysis and verification of security protocols can help to detect security flaws in the protocol design. Some of the flaws may be subtle and hard to find [97]. As designing security protocols is an error prone process, it is essential to formally verify these protocols to detect any subtle flaws which may not be found by informal security analysis. The public key protocol [98], designed by Needham-Schroeder, is a good example showing the importance of protocols formal verification. This protocol was considered as secure for years before it was formally verified by Lowe [99] [100] in 1995. Lowe verified this protocol using FDR (Failures Divergences Refinement) checker [101] with the CSP (Communicating Sequential Processes) logic [102], and discovered a flaw in the design of the protocol. Since this date, formal methods have been widely used for the formal verification of security protocols. Formal methods have been considered of crucial importance and have been the subject of intense research over the past decade. An overview of the formal methods is given below.

4.7.1 Formal Methods: An Overview

Generally, as illustrated in Figure 4.6, when a formal method is used to formally verify a security protocol, three steps are performed. In the first step, the protocol is formally modelled. In the second step, the security properties of the protocol
are formally specified. The first two steps are accomplished by using a high-level formal notation language. In the third step, using a tool (e.g. MOCHA in our case) that understands the outputs of step one and step two, the protocol modelled in step one is verified against its properties specified in step two. There are various tools that can be used to help with the verification process. Examples of these tools include theorem prover ISABELLE, finite-state analysers, Murφ, NRL, Model checker Spin, Model checker Mocha and FDR.

The literature contains many different formal methods, e.g. BAN belief logics, SVO logic, and inductive theorem proving. These methods are used to formally analyse different security protocols with various security properties. BAN (Burrows, Abadi and Needham) belief logic, among the first formal methods, has been used to formally verify authentication protocols, such as Kerberos and Otway-Rees. SVO (Syverson and van Oorschot) belief logic has been used to verify non-repudiation protocol. The inductive theorem proving method has also been used to formally analyse a number of well-known protocols. Examples include the Kerberos protocol, TLS protocol, e-commerce protocol, SET (Secure Electronic Transactions), and smart card protocols. Finite-state exploration is another formal method which has been used to analyse protocols such as Needham-Schroeder, Kerberos, and SSL.

The effort on formal verifications of security protocols have largely been focused on authentication and key-establishment properties. The same level of efforts has not been given to the fairness property. The nature of attacks on fair

Figure 4.6: General approach of formal methods
exchange protocols is different from that of the attacks on authentication protocols. Therefore, methods designed for the formal verification of authentication protocols are not readily applicable to fair exchange protocols. In the model of an authentication protocol, an attacker has control over the network, and protocol participants trust each other. For a fair exchange protocol, on the other hand, protocol participants do not trust each other, and any one of them may misbehave. As a result, a TTP is used, and only the TTP is assumed to be trusted and to follow the protocol specification faithfully. These differences in the protocol features should be reflected in the models of the respective protocols to be analysed, and in the formalisation of their security properties.

A number of different formal methods \[102\], \[108\], \[117\], \[118\] have been used to formally analyse fair exchange protocols. Schneider et al \[119\] used CSP to formally analyse a non-repudiation protocol. Zhou et al \[111\] applied SVO logic to analyse non-repudiation protocols. Shmatikov et al \[120\] applied the finite-state tool Murφ to analyse a fair exchange protocol. Kremer et al \[121\] used a temporal logic, ATL, with a game semantics and the corresponding model checker Mocha to analyse non-repudiation, fair exchange, and contract signing protocols.

As a special type of fair exchange protocols, contract signing protocols have been formally analysed with different verification tools. In \[122\], Chadha et al have used the multiset rewriting formalism together with inductive methods to analyse the abuse-free property in a contract signing protocol. The property is expressed in terms of strategies, which provide a natural framework for the analysis. In \[120\] and \[123\], Shmatikov and Mitchell applied the finite-state tool Murφ to analyse abuse-free contract signing protocols. In \[121\], Kremer et al used the finite model-checker Mocha to analyse two contract-signing protocols one of them is abuse-free. Spin model checker \[106\] is used to verify contract signing protocols in \[124\]. More recently, a Strand space \[125\] and Petri-nets \[126\] have been used to formally analyse contract signing protocols.

As described above, there are many tools to analyse the properties of contract signing protocols. However, not all of them are suitable to analyse the abuse-free property of these protocols. With the inductive method, the proof of the property has to be carried out by hand, which is a time-consuming process. Using Murφ, the protocol property is modelled as an invariant. This modelling method is not sound for an abuse-free property which cannot be expressed using invariants. Spin tool has not been used to analyse the abuse-freeness property. This is because,
as reported in [127], Spin uses LTL (Linear Temporal Logic) to formulate the properties of a fair exchange protocol but LTL cannot be used to express the abuse-freeness property. Strand space is a time-consuming process as it is a pen and paper tool.

Mocha is another tool which has been used to formally analyse contract signing protocols with the abuse-free property. As reported by Kremer et al [121], Mocha is advantaged over Murϕ in that Mocha allows the specification of protocol properties in ATL. ATL is a temporal logic with game semantics and its use allows a more natural expression of properties, especially the abuse-free property. Owing to this advantage, Mocha has recently been used to verify a number of multi-party contract signing protocols [128].

Based upon the above considerations, the finite model checker MOCHA and ATL have been chosen to formally verify our RDS protocol in this thesis.

4.7.2 Mocha Model Checker

The Mocha verification tool is an interactive verification software environment for the modular and hierarchical verification of heterogeneous systems [21], [129]. It is a software for specification, simulation, and verification. As depicted in Figure 4.7, Mocha uses Alternating Transition Systems (ATS) [117] to model protocols. ATS is a game based model. It uses a high level language called Guarded Command Language to describe protocols. Mocha also uses Alternating Temporal Logic (ATL) [117] to express requirements that the protocols must verify. ATL is a game based logic to reason about protocol properties. A Mocha model checker is then used to verify whether the requirements specified by ATL meet the protocol modelled as ATS system. This section gives a brief overview of the components of Mocha, i.e. ATS, ATL and its Guarded Command Language.

4.7.2.1 Alternating Transition System (ATS)

ATS is a game variant of usual Kripke structures. The first formal definition of ATS was given in [117]. It is used to model a protocol as a game. In this thesis, we only give an introduction about ATS. For more details, refer to [117]. An ATS system, $S$, is defined as a 6-tuple, i.e. $S = (\Pi, \Sigma, Q, Q_0, \pi, \delta)$, where:

- $\Pi$ is a finite set of propositions;
- $\Sigma$ is a finite set of players;
• $Q$ is a finite set of states;

• $Q_0 \subseteq Q$ is a set of initial states;

• $\pi : Q \rightarrow 2\Pi$ is a labelling function that labels states with propositions;

• $\sigma : Q \times \Sigma \rightarrow 2 \setminus 2Q\emptyset$ is a game transition function that maps a state and a player to a nonempty set of choices, where each choice is a set of possible next states. Furthermore, if $\Sigma = a_1, ..., a_n$, then for every state $q \subseteq Q$ and each possible $Q_1, ..., Q_n$ where $Q_i \subseteq \sigma(q, a_i)$, $Q_1 \cap ... \cap Q_n$ is a singleton.

For every player, $a$, and state, $q$, the game transition function defines the set of choices $\sigma(q, a) = Q_1, Q_2, ..., Q_n$, with $Q_i \subseteq Q$. This set of choices can be played by the player in the state $q$. A choice is a set of possible next states. One step of the game at a state, $q_i$, is played in the following way: each player $a \subseteq \Sigma$ makes his choice and the next state of the game $q'$ is the intersection (that is required to be a singleton) of the choices made by all the players of $\Sigma$, i.e. $q' = \cap a \subseteq \Sigma(q, a)$. A computation of a system $S$ is an infinite sequence, $\lambda = q_0q_1...q_n$, of states obtained by starting the game in $q_0$, where $q_0 \subseteq Q$. 
4.7. RDS PROTOCOL FORMAL VERIFICATION

4.7.2.2 Alternating-time Temporal Logic (ATL)

ATL is a logic in which one is able to reason about strategic abilities of players [117]. This means that ATL is the best logic to reason about an ATS model [121, 130, 131]. It is defined in terms of a finite set $\Pi$ of propositions and a finite set $\Sigma$ of players. Given the finite set $\Pi$ of propositions, and the finite set $\Sigma$ of players, an ATL formula could be of one of the following:

- $p$, where $p \in \Pi$, is a proposition.
- $\neg \varphi$ or $(\varphi_1 \lor \varphi_2)$, where $\varphi$, $\varphi_1$, and $\varphi_2$ are ATL formulae.
- $\neg << A >> \lozenge \varphi$, $<< A >> \circ \varphi_1$, $<< A >> \Box \varphi$, or $<< A >> \varphi_1 \lor \varphi_2$, where $A \subseteq \Sigma$ is a set of players, and $\varphi, \varphi_1$ and $\varphi_2$ are ATL formulae.

The operators used in the above formulae are defined as follows:

- $<< >>$ is a path quantifier, and
- $\circ, \Box, \lozenge$, and $U$ are temporal logic operators, where $\circ$ denotes “next”, $\Box$ refers to “always”, $\lozenge$ denotes “eventually” and $U$ refers to “until”.
- $\neg$, and $\land$ are logical connectives that have the standard interpretation, i.e. $\neg$ denotes “negation”, and $\land$ refers to “or”.

ATL logic enables the modelling of both cooperative and adversarial behaviours amongst the players. We use the following example to show its expressive power. For a set of players, $\Sigma = a, b, c$, consider the verbal reading of the following formulae:

- $<< c >> \lozenge p$, this formula means that the player, $c$, has a strategy against both players, $a$ and $b$, to eventually reach a state where the proposition, $p$, is true;
- $\neg << b, c >> \Box p$, this formula means that the coalition of the players, $b$ and $c$, does not have a strategy against the player, $a$, to reach a point where the proposition, $p$, will always be true;
- $<< a, b >> \circ (p \land \neg << c >> \Box p)$, this formula says that the players, $a$ and $b$, can cooperate so that the next state satisfies $p$, and from this state, the player, $c$, does not have a strategy to impose $p$ forever.
The above formulae illustrate how ATL can be used to express both cooperative and adversarial behaviours among players. These cooperative and adversarial behaviours are typical behaviours of a contract signing protocol (e.g. our RDS protocol). To determine which player can win a game, ATL formulae (e.g. $<<A>>\psi$ at a state, $q$, of the ATS system, $S$) of this game need to be evaluated. The idea of how to evaluate such ATL formula can be illustrated by the following example. Consider a two-player game between a protagonist and an antagonist with the following specifications. A set of players, $A \subseteq \sigma$, a set of computations $\Lambda$, and a state, $q$, which is the starting state of the game and let the game be in a position $y$, at each step of the game, to determine the next state, the protagonist first chooses, for every player, $a \in A$, a set $Q_a \in \delta(y,a)$. Then, the antagonist chooses a successor, $z$, of $y$ such that $z \in Q_a$ for all $a \in A$, and the game position is updated to $z$. Following the same way, the game between a protagonist and an antagonist continues forever and produces a computation. If the resulting computation satisfies the formula, $<<A>>\psi$, then the protagonist wins the game. In other words, if the ATL formula $<<A>>\psi$ holds at the state $q$, it means that the protagonist has a strategy to win this game. Note that, $\psi$ is a linear temporal formula whose outermost operator is $\diamond$, $\square$, $\Diamond$, or $U$.

### 4.7.2.3 Guarded Command Language

Using ATS notations to directly model a protocol (e.g. RDS protocol) is difficult as it is not easy to understand. A more user oriented language, called the *Guarded Command Language*, can be used to model the protocol as ATS system. This modelling language is part of the Reactive Modules language [600] which is part of the model checker Mocha. More details about the syntax and semantics of this language can be found in [13]. Here, we give some description about the Guarded Command Language.

In an ATS system, a description of a player, $a \in \sigma$, is represented by a set of guarded commands of the following form:

$$[\text{guard}_\varsigma \rightarrow \text{command}_\varsigma]$$

where,

- *Guard* is a Boolean predicate over state variables;
- *Command* is an update predicate.
Each player, \( a \in \sigma \), selects one of his commands whose Boolean guard evaluates to \( true \). The next state is then obtained by taking the conjunction of the effects of each updated part of the commands selected by the players.

Figure 4.8 shows a sample of guarded commands for players \( A, B, \) and \( C \). To illustrate these commands, the following syntactic assumptions are used. Firstly, when updating state of a command, if a variable, \( p \), does not appear in the right-hand side of the command, it is considered as unchanged by the command. Hence, implicitly, we have \( p = p' \) for each such variable. For example, in the command \( ([]q \rightarrow r' := true) \), \( q \) does not show in the right-hand side. Thus, the value of \( q' \), in this case, will be the same as the value of \( q \). Secondly, the command, \( true \rightarrow \), means that the player owning this command can choose to leave its controlled variables unchanged. This is considered as “idle” action. Note that as the guard is \( true \), that action can be taken at any step of the game if no fairness constraints are imposed on the player.

Now the explanation of the commands in Figure 4.8 is as follows. The command \( ([]true \rightarrow p' := true) \) means that \( p \) is set to \( true \) as its guard is always \( true \). The command \( ([]p \rightarrow q' := true) \) says that when \( p \) is \( true \), \( q \) will be \( true \). The command \( ([]q \rightarrow r' := true) \) means that when \( q \) is \( true \) \( r \) will be also \( true \).

### Figure 4.8: A simple program written by the Game Guarded Command language

<table>
<thead>
<tr>
<th>A: // player A has the following guarded commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>[] true ( \rightarrow ) p:=true</td>
</tr>
<tr>
<td>[] true ( \rightarrow )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B: // player B has the following guarded commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>[] p ( \rightarrow ) q:=true</td>
</tr>
<tr>
<td>[] true ( \rightarrow )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C: // player C has the following guarded commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>[] q ( \rightarrow ) r:=true</td>
</tr>
<tr>
<td>[] true ( \rightarrow )</td>
</tr>
</tbody>
</table>

### 4.7.3 Model Checker and RDS Protocol Modelling

The following assumptions are used in the modeling of the RDS protocol.

1. Only valid protocol messages can be sent.
2. Each protocol execution is uniquely identified.
3. Cryptanalysis is not considered in this model.

When choosing a model checker for a given protocol, one should consider the properties of this protocol. There are two differences between modelling classical security protocols, e.g. authentication and key-exchange protocols and contract signing protocols. The first difference is that a classical security protocol typically consists of only one protocol aimed at authentication and secrecy. On the other hand, a contract signing protocol usually consists of more than one sub-protocol that can be executed at any time. Executing a signing protocol at a time, which is not anticipated by its designer, may cause unexpected errors. These errors may be utilized by a signer to gain an advantage over the other signer. The second difference is that a classical security protocol is usually designed to be secure against external intruders while a contract-signing protocols is designed to be secure against malicious insiders, i.e. signers. Therefore, when modelling a contract signing protocol (e.g. an RDS protocol) two processes should be modelled, one for a honest signer, and the other for a dishonest signer. The first process describes an honest behaviour of a signer while the second process describes a dishonest behaviour of the same signer.

A contract signing protocol is modelled as a game and its participants as players. In the modelling of the RDS protocol, there are four players: Alice, Bob, LI, and the communication channels. An instance of the RDS protocol will first be modelled as an ATS system. Each protocol participant will then be modelled as a player in the ATS system using the guarded command language just described above. As illustrated in Figure 4.9 the verification process of the RDS protocol comprises three steps:

1. Modeling the protocol using the guarded command language (i.e. creating a ATS system for the RDS protocol);
2. Translating protocol properties, fairness, non-repudiation, and abuse-freeness into ATL formulae; and
3. Running the model produced in the first step and the specification produced in the second step using the MOCHA verification tool.

### 4.7.3.1 Modelling the RDS Protocol

This section describes the modelling of the players of the RDS protocol, i.e. Communication channels, LI, Bob, and Alice.
4.7. **RDS PROTOCOL FORMAL VERIFICATION**

**Communication Channels**: Assuming that the channels between every pair of the involved players are resilient (i.e. assumption 7 given in Section 4.4.2), the communication channels are modelled using shared variables. The messages transmitted through these channels are modelled by three steps. In the first step, sending of a message, $Msg_1$, through a channel, $Chann$, is modelled by setting a variable called, $SendMsg_1$, to true. In the second step, the delivery of $Msg_1$ is modelled by setting another variable called, $Msg_1$, to true. In the third step, a guarded command representing the transmission is as follows: $\langle[] SendMsg_1 \rightarrow Msg_1 := true \rangle$. This means that when $SendMsg_1$ is true, then $Msg_1$ will be true as well.

```plaintext
/* Communication channel between LI and Bob */
1. $\langle[] SendRD_Act_Req6 \& \neg RD_Act_Req6 \rightarrow RD_Act_Req6 := true \rangle$
2. $\langle[] SendPositive_Result_B \& \neg Positive_Result_B \rightarrow Positive_Result_B := true \rangle$
```

Figure 4.10: A segment of the communication channel model

Figure 4.10 gives part of the guarded command describing the communication channel between LI and Bob. The first guarded command models that a channel
delivers the message \textit{RD\_Act\_Req6} to LI. The second command expresses that the channel delivers the message \textit{Positive\_Result\_B} to Bob.

**License Issuer (LI)**

In the RDS protocol, as LI is a trusted entity, it is considered as a special player. In other words, LI is not biased. It does not have a strategy to help any of the players to cheat the other. LI should be modelled in a deterministic way. That is, at each stage of the RDS protocol execution, LI follows the protocol specification faithfully, and only performs the actions defined in the protocol design.

![Segment of LI's model](image)

Figure 4.11: A segment of LI's model

Figure 4.11 shows a sample of guarded commands describing LI's model. The first command expresses that LI has received the message, \textit{RD\_Act\_Req6} from Bob. This means that once LI obtains the message, \textit{RD\_Act\_Req6}, it will have knowledge of the items, \textit{(ASign\_B, A, A\_L, RD\_L, Payment\_B, RP\_Lic)}.

The second command describes what LI performs when it accepts the RD: sending \textit{ks} and \textit{ASign\_B} to Bob and Alice, respectively.

**Alice and Bob**

There are two ways of modelling Alice and Bob. The first way models both Alice and Bob as honest players, i.e. both players execute the protocol honestly. The second way models either Alice or Bob as a malicious player. Hereafter, we will use \textit{A} and \textit{B} to refer to honest Alice and honest Bob, respectively, and \textit{Dis\_A} and \textit{Dis\_B} denote dishonest Alice and dishonest Bob, respectively. Each player, \textit{A}, \textit{Dis\_A}, \textit{B}, and \textit{Dis\_B} is represented by a set of guarded commands.

**Modelling A and B:** To model honest players, \textit{A} and \textit{B}, four actions are described: \textit{initial state, creating messages, sending messages, and receiving messages}). The initial state represents the initial knowledge of each player. For example, the initial knowledge of the player, \textit{A}, includes \textit{ks}, \textit{f}, and \textit{RD contract}. 
These items are modelled as Boolean variables, and they are initialized to \textit{true}. Other variables (e.g. $\text{SendMsg}_1$, $\text{SendMsg}_2$) are set to \textit{false} to indicate that their values are not known prior to the protocol execution.

The following method is used to model the creation of a protocol message. Assuming that entity $A$ creates a message, $\text{Msg}_i$, which consists of $N$ items, i.e., $\text{Msg}_i = \text{item}_{i1}, \text{item}_{i2}, ..., \text{item}_{in}$. The creation of $\text{Msg}_i$ is modelled by adding a guarded command, $\text{item}_{i1} \land \text{item}_{i2} \land ... \land \text{item}_{in} \rightarrow \text{Msg}_i := \text{true}$, to $A$’s description.

Sending a message is another action taken by a player. This action is modelled in the following way. Assuming that $A$ sends $B$ a message, $\text{Msg}_i$, i.e. $A \rightarrow B : \text{Msg}_i = \text{item}_{i1}, \text{item}_{i2}, ..., \text{item}_{in}$. Modelling this action is done by adding the guarded command, $\text{item}_{i1} \land \text{item}_{i2} \land ... \land \text{item}_{in} \rightarrow \text{SendMsg}_i := \text{true}$, to $A$’s description. This guarded command consists of the conjunction of all the elements of the message. This means that $A$ should only send a message \textit{if and only if} he has got knowledge of all the required items.

The transmission of this message is described by another guarded command, $\text{SendMsg}_i \rightarrow \text{Msg}_i := \text{true}$. This guarded command is added to the description of the communication channel between $A$ and $B$. In the case of delaying a message, the communication channel executes an idle guarded command, i.e., $(\text{true} \rightarrow)$, which is added to the player’s description. This is explained in Figure 4.8.

To model the reception of a message, $\text{Msg}_i$, by $B$, the following guarded command is added to $B$’s description, $\text{Msg}_i \rightarrow \text{item}_{i1} := \text{true}; ...; \text{item}_{in} := \text{true}$. These $N$ items are set to \text{true} for the first time in $B$’s description. $B$ could now use the items set to \text{true} to create subsequent protocol messages.

```plaintext
/* Modeling Honest Alice */
/* Alice sends Msg1 to Bob */
1- [] RD & RP_Lic & ASign_A & ~Stop_A & ~SendMsg1 -> SendMsg1 := true

/* Alice has received Msg2 from Bob, i.e. she will receive Asign-Ba */
2- [] Msg2 & ~Stop_A & ~Asign_Ba -> Asign_Ba := true
```

Figure 4.12: An example of guarded commands of $A$’s description

Figure 4.12 shows part of $A$’s description in the guarded commands. The first command expresses that “\textit{if Alice has knowledge of (RD, RP\_Lic and ASign\_A) and Alice hasn’t stopped the RDS protocol execution, then she can send $\text{Msg}_1$}”. The second command models the reception of $\text{Msg}_2$ by $A$. It describes that “If
Alice has got $Msg_2$, and Alice hasn’t stopped the RDS protocol execution, she can receive Bob’s signature, $ASign_{Ba}$.

Figure 4.13: Part of guarded commands of B’s description

Figure [4.13] illustrates part of guarded commands modelling $B$. The first guarded command models the reception of $Msg_1$ by Bob. In other words, it says that “once Bob obtains $Msg_1$, he gets (RD$_b$, RP$_{Lic}_b$, and $ASign_{A_b}$)”. This command means that Bob increases his knowledge once he receives $Msg_1$. The second and the third commands, respectively, describe that Bob performs verifications $B_v1$ and $B_v2$ after receiving $Msg_1$.

**Modelling Dis$_A$, and Dis$_B$:** In modelling dishonest players, Dis$_A$ and Dis$_B$), the guarded commands of Dis$_A$ and Dis$_B$ are obtained from their honest counterparts, A and B. This is done by applying two operations: (1) relax any predefined order of executing A’s and B’s messages; and (2) add new actions (guards) that describe malicious behaviours. Figure [4.14] illustrates part of Dis$_A$’s description. The first command in Figure [4.14] is the same as the second command shown in Figure [4.12]. The second command in Figure [4.14] expresses that Alice’s misbehaviour (prematurely stop the protocol execution) is added to Dis$_A$’s model. It says that “once Dis$_A$ has got $Msg_2$, she prematurely stops the RDS protocol execution”.

Figure 4.14: Part of guarded commands describing the player, Dis$_A$
4.7. RDS PROTOCOL FORMAL VERIFICATION

Similar to the description of Dis_A, Figure 4.15 shows part of the description of Dis_B. Most of the commands here describe the honest B, except that the second command is added to express Bob’s misbehaviour. This command models that “once Dis_B has received Msg_1, he sends LI an incomplete RD activation request”.

```
/* Modeling Dis_Honest Bob */
/* Dis_B receives Msg_1 from Alice */
1- [] Msg_1 & ~Asign_Ab & ~Stop_B -> RD_b'=true ; RP_Lic_b'=true ; Asign_Ab'=true

/* Upon the receipt of Msg_1, Dis_B sends LI an incomplete RD_Act_REQ. 
( ex. excluding payment) */
2- [] Msg_1 & Asign_Ab & RD_b & RP_Lic_b & ~Stop_B & ~SendRD_Act_Req1 -> endRD_Act_Req1 := true
```

Figure 4.15: Part of Dis_A’ description using guarded commands

Protocol messages: The messages of the players A, B, Dis_A, and Dis_B are modelled in the following manner. All the protocol messages are modelled as Boolean variables, which are initialized to false and then set to true when they are sent by their senders. Signatures, Asign_A and Asign_B, keys and RD contracts are also modelled as Boolean variables, which are initialised as false and updated by their senders during the course of the protocol execution. We model the action of sending out signatures, or other messages as guarded commands in which the senders reset the corresponding variables to true. The modelling of the reception of messages is done by making their receivers reset the corresponding variables to true.

4.7.3.2 Modelling the Properties of the RDS Protocol

The second step of the verification process, as shown in Figure 4.7, is to specify the security properties of the RDS protocol, namely, fairness, non-repudiation, and abuse-freeness. Using the ATL formulae described in Section 4.7.2.2 these properties are expressed as strategies.

Note that, in the following properties specification, the proof of a signed RD for Bob is of the form (Asign_A & (ks v ks_LI)) and the proof of a signed RD for Alice is of the form (Asign_B v Asign_B_LI).

Fairness

A protocol is said to be fair if, at the end of the protocol execution, either both
Alice and Bob obtain a signed contract or none of them gets anything useful. This
definition can be split into two parts expressing fairness for Alice and fairness
for Bob.

**Fairness for Alice:** The RDS protocol is fair for Alice if “a coalition of
Dis_Bob and the communication channels do not have a strategy to reach a state
where Dis_Bob can get his copy of the signed RD but Alice has no strategy to
obtain her copy of the signed RD”. This definition can be modelled by the ATL
formula

\[ \neg \langle\langle \text{Dis}_\text{Bob}, \text{Chann} \rangle\rangle \diamond ((\text{ASign}_\text{A} \land (\text{ks} \lor \text{ks}_{\text{LI}}))) \land \\
\neg \langle\langle \text{Alice} \rangle\rangle \diamond (\text{ASign}_\text{B} \lor \text{ASign}_\text{B}_{\text{LI}}) \]  

(4.1)

Here, *Chann* denotes the communication channels, \((\text{ASign}_\text{A} \land (\text{ks} \lor \text{ks}_{\text{LI}}))\) denotes that Alice has generated \(\text{ASign}_\text{A}\) (i.e. has signed the RD) and \(\text{ks}\) and \(\text{ks}_{\text{LI}}\) is the keystone sent either by Alice or by LI, respectively. \((\text{ASign}_\text{B} \lor \text{ASign}_\text{B}_{\text{LI}})\) refers to the proof that Bob has also signed the RD where \((\text{ASign}_\text{B} \lor \text{ASign}_\text{B}_{\text{LI}})\) is Bob’s ambiguous signature sent either by Bob or by LI, respectively. In addition, the operator \(\diamond\) is an ATL temporal logic operator which means ”eventually” while the operators, \(\neg, \land, \lor, \Rightarrow\), are standard logic operations.

**Fairness for Bob:** The RDS protocol is said to be fair for Bob if ”Dis_Alice
in collaboration with the communication channels do not have a strategy to reach
a state where Dis_Alice can get her copy of the signed RD but Bob has no a
strategy to obtain his copy of the signed RD”. The ATL formula (4.2) expresses
the fairness for Bob:

\[ \neg \langle\langle \text{Dis}_\text{Alice}, \text{Chann} \rangle\rangle \diamond ((\text{ASign}_\text{B} \lor \text{ASign}_\text{B}_{\text{LI}})) \land \\
\neg \langle\langle \text{Bob} \rangle\rangle \diamond (\text{ASign}_\text{A} \land (\text{ks} \lor \text{ks}_{\text{LI}}))) \]  

(4.2)

It is worth noting that with the ATL formulae (4.1) and (4.2), (a) Dis_Bob
(Dis_Alice) has been given all the power necessary for Dis_Bob (Dis_Alice) to
cheat Alice (Bob), and (b) we did not add \(\text{ks}\) to \((\text{ASign}_\text{B} \lor \text{ASign}_\text{B}_{\text{LI}})\) as
Alice has already got \(\text{ks}\) before the RDS protocol run is started.

**Non-Repudiation**

The non-repudiation property is the ability to make sure a sender of a message
cannot repudiate the sending of the message and to ensure that a receipt of a
4.7. RDS PROTOCOL FORMAL VERIFICATION

message cannot deny having received the message \([132]\). This property has two sub-properties: *Non-repudiation of Origin (NOO)* and *Non-repudiation of Receipt (NOR)*. NOO, in our context, means that the recipient of a signature is assured that the signature is indeed generated by the claimed signer. NOR means that the sender of a signature is provided with evidence that the intended recipient has indeed received this signature.

**Non-repudiation of Origin (NOO):** In the context of the RDS protocol, there are two NOOs: \(NOO_{ASign_A}\) and \(NOO_{ASign_B}\). \(NOO_{ASign_A}\) is a token which proves to Bob that Alice has indeed created \(ASign_A\). The token contains either \((ASign_A&ks)\) or \((ASign_A&ks_{LI})\). In other words, when Bob receives both \(ASign_A\) and \(ks\) from Alice, the NOO token is \(NOO_{ASign_A} = (ASign_A&ks)\). However, if Bob receives \(ASign_A\) from Alice, but \(ks_{LI}\) from LI, then the NOO token is \(NOO_{ASign_A} = (ASign_A&ks_{LI})\). This can be informally expressed in the formula (4.3) as:

\[
NOO_{ASign_A} = Bob.knows(ASign_A) \land (Bob.knows(ks) \lor Bob.knows(ks_{LI})) \tag{4.3}
\]

Using ATL, \(NOO_{ASign_A}\) can be formally described as:

\[
<< Bob >> \Box (ASign_A \land (ks \lor ks_{LI})) \tag{4.4}
\]

The above formula says that upon the successful completion of an RDS protocol run, Bob will obtain \((ASign_A&ks)\) or \((ASign_A&ks_{LI})\). Either of these two tokens serves as NOO of Alice’s signature.

The other NOO evidence of the RDS protocol is \(NOO_{ASign_B}\), which is a token proving to Alice that Bob has indeed created \(ASign_B\). The token contains either \(ASign_B\) or \(ASign_B_{LI}\). If Alice receives \(Msg_2\) from Bob, then the token is \(NOO_{ASign_B} = ASign_B\). Otherwise, \(NOO_{ASign_B} = ASign_B_{LI}\) which is Bob’s signature sent by LI during the RD activation process. Informally, \(NOO_{ASign_B}\) can be written as shown in the equation (4.5):

\[
NOO_{ASign_B} = (Alice.knows(ASign_B) \lor Alice.knows(ASign_B_{LI})) \tag{4.5}
\]
Using ATL, $\text{NOO}_{\text{ASign}_B}$ can be formally described as (4.6):

\[
<< \text{Alice} >> \Box (\text{ASign}_B \lor \text{ASign}_BLI)
\] (4.6)

Formula (4.6) says that upon the successful completion of the RDS protocol, Alice will obtain $\text{ASign}_B$ or $\text{ASign}_BLI$. Either of these tokens serves as NOO of Bob’s signature. Note that Alice always holds the keystone, $ks$, which can be used along with $\text{ASign}_B$ or $\text{ASign}_BLI$ to form the NOO of Bob’s signature.

**Non-repudiation of Receipt (NOR):** Similar to the case of NOO, in the context of the RDS protocol, there are two NORs, $\text{NOR}_{\text{ASign}_A}$ and $\text{NOR}_{\text{ASign}_B}$. $\text{NOR}_{\text{ASign}_A}$ is to prove to Alice that Bob has indeed received $\text{ASign}_A$. The token contains either $\text{ASign}_B$ or $\text{ASign}_BLI$. If Alice receives $\text{ASign}_B$ from Bob in $Msg_2$, then the token will be $\text{NOR}_{\text{ASign}_A} = \text{ASign}_B$. This is because it is hard for Bob to generate $\text{ASign}_B$ without receiving $\text{ASign}_A$ in $Msg_1$, i.e. if Alice receives $\text{ASign}_B$ from Bob, then Bob must have received $\text{ASign}_A$. If Alice receives $\text{ASign}_BLI$ from LI after a successful RD activation process, the NOR token will be $\text{NOR}_{\text{ASign}_A} = \text{ASign}_BLI$. Since LI only sends Alice $\text{ASign}_BLI$ if LI receives from Bob a valid RD activation request that contains $\text{ASign}_B$, and $\text{ASign}_B$ already contains $\text{ASign}_A$. Thus, if Alice has got $\text{ASign}_BLI$, Bob must have got $\text{ASign}_A$.

$\text{NOR}_{\text{ASign}_A}$ can be informally written as:

\[
\text{NOO}_{\text{ASign}_A} = (\text{Bob knows (ASign}_B) \lor \text{Bob knows (ASign}_BLI))
\] (4.7)

To formally verify $\text{NOO}_{\text{ASign}_A}$ by Mocha, it should be written in ATL as:

\[
<< \text{Alice} >> \Box (\text{ASign}_B \lor \text{ASign}_BLI)
\] (4.8)

Formula (4.8) means that, upon the successful completion of a RDS protocol execution, Alice will obtain either $\text{ASign}_B$ or $\text{ASign}_BLI$. Either of these two items serves as NOR of $\text{ASign}_A$ for Alice.

The second NOO token is $\text{NOO}_{\text{ASign}_B}$ which proves to Bob that Alice has indeed received $\text{ASign}_B$. The token contains either $ks$ or $ksLI$. If Bob receives $ks$ from Alice in $Msg_3$, the token will be $\text{NOR}_{\text{ASign}_B} = ks$. If Bob gets $ksLI$ from LI after a successful RD activation process, $ksLI$ serves as $\text{NOO}_{\text{ASign}_B}$ for
4.7. RDS PROTOCOL FORMAL VERIFICATION

Bob, i.e. $NOO_{A_{signB}} = ks.LI$. This can be informally described as:

$$NOO_{A_{signB}} = (Alice.knows(ks) \lor Alice.knows(ks.LI))$$ (4.9)

To formally verify $NOO_{A_{signB}}$ by Mocha, the token should be written in ATL formula as:

$$<< Bob >> \Box (ks \lor ks.LI)$$ (4.10)

Formula (4.10) says that upon the successful completion of a RDS protocol run, Bob will obtain either $ks$ or $ks.LI$, either of which serves as an NOR of $A_{signB}$ for Bob.

**Abuse-freeness**

The RDS protocol is said to be abuse-free if, at any stage of a protocol execution, it is impossible for any signer, say Alice, to be able to prove to an outside party that Alice is in control of the outcome of the protocol execution. To formalize the abuse-freeness property, two issues should be considered [121]: (a) balance, and (b) abuse-free. The balance is a signer’s power to unilaterally decide the outcome of a protocol run, and the abuse-freeness is the signer’s ability to demonstrate this power to an outside party. The issue of the balance has been analysed in [120] and [123]. In [121] Kremer et al have proved that when a protocol is balanced, it is also abuse-free. Kremer et al also proposed a new definition for the abuse-freeness. In this definition, the abuse-freeness is split into two parts expressing abuse-freeness for Alice and for Bob, respectively. With this definition, a contract signing protocol is said to be abuse-free for Bob “if Alice cannot reach a state in the protocol execution, where Alice can prove to Charlie (a third party) that the protocol has been initiated with Bob, without Bob having a strategy to successfully complete the protocol”. Abuse-freeness for Alice is defined in the same way.

In the following part, we model the abuse-free property for Alice and for Bob in the context of the RDS protocol based on Kremer’s definition of the abuse-freeness.

**Abuse-freeness for Bob:** We say that the RDS protocol is abuse-free for
CHAPTER 4. A RESELLING DEAL SIGNING (RDS) PROTOCOL

Bob if the following ATL formula holds.

\[
\sim <<\text{Dis}_\text{Alice} >> \Diamond (\text{Dis}_\text{Alice}.\text{Prove}_{2TP} \land \\
\neg <<\text{Bob} >> \Diamond (ks \lor ks.LI))
\]  

(4.11)

Formula (4.11) says that \text{Dis}_\text{Alice} can not reach a state in a RDS protocol execution, where she can prove to a third party (e.g. another buyer) that the protocol has been initiated with Bob, without Bob having a strategy to successfully complete the protocol run. Note that, in the context of the RDS protocol, Bob can only successfully complete an RDS protocol execution under two cases: (1) if Bob can get the keystone, \( ks \), from Alice (i.e. receives \( ks \) in \( Msg_3 \)); (2) if Bob obtains the keystone, \( ks.LI=ks \), from LI after a successful RD activation process.

In formula (4.11), \text{Dis}_\text{Alice}.\text{Prove}_{2TP} is a predicate representing the ability of Alice to prove to a TP (Third Party) whether Bob has indeed signed RD. This predicate can be defined as follows:

\[
\text{Dis}_\text{Alice}.\text{Prove}_{2TP} = \text{Dis}_\text{Alice}.\text{Knows}(Msg_2)
\]

(4.12)

If this predicate is true, it means that \text{Dis}_\text{Alice} has indeed received \( Msg_2 \). Consequently, \text{Dis}_\text{Alice} can prove to a TP (e.g. Charlie) that Bob has indeed been involved in the RDS protocol execution. If this predicate is false, \text{Dis}_\text{Alice} cannot prove to Charlie that Bob has indeed started the RDS protocol execution.

**Abuse-free for Alice:** The abuse-free property for Alice can be modelled in the same manner as the case of Bob. We say that the RDS protocol is abuse-free for Alice if ATL formula (4.13) holds.

\[
\sim <<\text{Dis}_\text{Bob} >> \Diamond (\text{Dis}_\text{Bob}.\text{Prove}_{2TP} \land \\
\neg <<\text{Alice} >> \Diamond (\text{ASign}_B \lor \text{ASign}_B.LI))
\]

(4.13)

Formula (4.13) says that \text{Dis}_\text{Bob} cannot reach to a state in an RDS protocol run, where \text{Dis}_\text{Bob} can prove to a third party (e.g. Sally) that the protocol execution has been initiated with Alice, without Alice having a strategy to successfully complete the protocol run. Note that, in the context of the RDS protocol, Alice can only successfully complete an RDS protocol run under two cases: (1) Alice gets \text{ASign}_B from Bob (i.e. receives \text{ASign}_B in \text{Msg}_2); (2) Alice obtains \text{ASign}_B.LI = \text{ASign}_B from LI after a successful RD activation process.
In formula (4.13), $\text{Dis}_{Bob}.Prove_{2TP}$ is a predicate representing the ability of Bob to prove to a TP whether Alice has indeed engaged with the RDS protocol run. This predicate can be defined as follows:

$$
\text{Dis}_{Bob}.Prove_{2TP} = \text{Dis}_{Bob}.Knows(Msg_3)
$$

If this predicate is true, it means that $\text{Dis}_{Bob}$ has indeed received $Msg_3$. Consequently, $\text{Dis}_{Bob}$ can prove to a TP (e.g. Sally) that Alice has certainly involved in an execution of an RDS protocol. If it is false, $\text{Dis}_{Bob}$ cannot prove to Sally that Alice has indeed involved in the RDS protocol run with Bob.

### 4.7.3.3 RDS Verification Using Mocha

The third step of the formal verification process is to feed the ATL formulae described in Section 4.7.3.1 to the Mocha model checker. Mocha then starts verifying each formula to check whether it is satisfied in all possible moves of the model players. Figure 4.16 shows a screenshot of Mocha while verifying the RDS protocol. This section, further, discusses the results obtained from the formal verification of the RDS protocol using Mocha.

**Fairness**

The RDS protocol is said to be fair if upon the completion of a protocol execution or upon an acceptance of an RD activation request by LI, either both the signers are committed to the contract RD or neither is. To verify this property, we have applied formulae (4.1) and (4.2) to Mocha. The outcome of the verification has shown that the RDS protocol is fair.

**Non-repudiation**

The RDS protocol is said to preserve the non-repudiation property if both NOO and NOR are supported. To verify that NOO is supported, we have verified the ATL formulae (4.4) and (4.6) with Mocha. NOR is also verified by checking the ATL formulae (4.8) and (4.10) with Mocha. The results of these verifications have shown that the RDS protocol satisfies both NOO and NOR properties.

**Abuse-free**

To check the abuse-free property of the RDS protocol, we have applied the ATL formulae (4.13) and (4.11) to Mocha. The verification of this property has led to the identification of two attacks on the abuse-free property. These attacks
and their countermeasures are discussed below.

**Attack 1 on Abuse-freeness** (AoAF1): As illustrated in Figure (4.17), the first attack could be mounted by Alice after she receives Msg2 from Bob. As Alice has knowledge of the keystone, ks, once she receives Bob’s signature in Msg2, Alice can determine the outcome of the RDS protocol run and prove to a TP that Bob has indeed signed the RD. This attack is further illustrated by the following scenario.

As depicted in Figure (4.17), to sign a deal, RD, with Bob, Alice first creates and sends Msg1 to Bob. On the receipt of Msg1, Bob performs verifications $B_{e1}$ and $B_{e2}$. If they are positive, Bob creates and sends Msg2 (i.e. $ASign_B$) to Alice.
4.7. RDS PROTOCOL FORMAL VERIFICATION

After receiving $Msg_2$, Alice performs verifications, $A_{v1}$ and $A_{v2}$. If they are both positive, it means that Bob is now committed to the RD. Now, Alice has the ability to determine the outcome of the RDS protocol run. For example, Alice could not send Bob $ks$ contained in $Msg_3$. Therefore, Bob will not be able to make Alice bind to the RD, while Alice can. Alice can also use Bob’s signature, $A_{Sign_B}$, received in $Msg_2$ along with $ks$ to prove to a third party (e.g. Charlie) that Bob has certainly signed the RD. This is because Alice is in control of $ks$ which makes $A_{Sign_B}$ binding to Bob. Therefore, Alice can abuse Bob’s signature.

The following scenario explains how Alice uses $AoAF_1$ to cheat Bob. Alice can delay sending $Msg_3$, i.e. $ks$, to Bob and prove to Charlie that Bob has indeed signed the RD deal. As illustrated in Figure 4.17 during the period (between receiving $Msg_2$ from and sending $Msg_3$ to Bob) Charlie may be desperately looking for a second-hand license, $Lic$. Charlie can then engage with Alice to sign another deal for $Lic$, i.e., $RD_2$. Once Charlie receives $Msg'_1$ from Alice, Charlie

Figure 4.17: Schematic figure of the $AoAF_1$ on the abuse-freeness property.
generates an RD\textsubscript{2} activation request (RD\textsubscript{2}−Act−Req) and sends it to LI. Alice then sends Msg\textsubscript{3} to Bob who performs BV\textsubscript{4} and then generates an RD activation request (RD−Act−Req) and sends it to LI. Bob’s RD−Act−Req, however, will be rejected by LI, as LI has already received Charlie’s RD\textsubscript{2}−Act−Req. In this scenario, Bob’s signature has been abused. This means that the RDS protocol is not abuse-free.

**Countermeasure to AoAF\textsubscript{1}**: The design of the RDS protocol has taken the following measures to counter the AoAF\textsubscript{1} attack.

**Measure 1**: Allowing Bob to send an RD activation request to LI once Bob receives Msg\textsubscript{1};

**Measure 2**: Allowing Bob to send Alice Msg\textsubscript{2} and then also send an RD activation request to LI;

**Measure 3**: Send Alice Msg\textsubscript{2} and then signs another RD\textsubscript{3} with another reseller (Sally), so Bob can choose either to activate RD or RD\textsubscript{3}.

Using one of these methods, the RDS protocol puts the control of accepting or rejecting the RD in the hands of Bob. In other words, as depicted in Figure 4.18 these measures will allow Bob to choose from the following options:

**Option 1**: if Bob has chosen Measure 1, he can avoid the AoAF\textsubscript{1} attack as Bob will send his RD activation request to LI before Alice even starts another RD signing protocol with Charlie

**Option 2**: if Bob has chosen Measure 2, Bob will counter the AoAF\textsubscript{1} attack as he will send LI his RD activation request before Charlie does.

**Option 3**: if Bob has selected Measure 3, he will not wait for Msg\textsubscript{3} from Alice. Rather, he will directly engage with another reseller (Sally) to sign another deal (RD\textsubscript{3}). Thus, during the period between sending Msg\textsubscript{2} and receiving Msg\textsubscript{3}, Bob may get another RD\textsubscript{3} which could be better than RD being signed with Alice. In this case, Bob will get two deals (RD and RD\textsubscript{3}), and he can choose the better one of the two to activate. Therefore, if Alice mounts the AoAF − 1 attack, Bob will not be affected.

To further illustrate the consequences of the AoAF − 1 attack, we summarise who gains and who loses in various scenarios in the following table:
Table 4.1: Actions and consequences of delaying $Msg_2$

<table>
<thead>
<tr>
<th>Actions</th>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Bob waited and got ks from Alice</td>
<td>Nothing</td>
<td>Signed RD that may be activated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Bob waited and got ks from Alice. He also signed a better $RD_3$ with another reseller</td>
<td>Nothing</td>
<td>Signed RD that may be activated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 4.1 – continued from previous page

<table>
<thead>
<tr>
<th>Actions</th>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>loses</td>
<td>gains</td>
<td>loses</td>
</tr>
<tr>
<td>4. Bob waited for <em>ks</em> from Alice but he never got it. He has also got another better deal, <em>RD₃</em>, signed with another reseller. On the other hand, Alice may have signed another deal, <em>RD₂</em>, with another buyer.</td>
<td>Nothing</td>
<td>Waiting forever: As Alice has got a better deal, <em>RD₂</em>, she may never send Bob the keystone, <em>ks</em>. In addition, since Bob did not sign any other deal, he may wait for <em>ks</em> that never arrives.</td>
</tr>
<tr>
<td>Two signed <em>RD₈</em>, <em>RD</em> and <em>RD₂</em> which is better than <em>RD</em>. If <em>RD₂</em> is activated, Alice can gain benefits.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
From Table 4.1, we can make two remarks. Firstly, if Alice refused to send \( ks \) to Bob, Bob has three options, \textit{Option 1-to-Option 3}, to overcome Alice’s misbehaviour. Secondly, there is one case in which Bob may be affected. This case happens if Bob wants to collect a full-signed RD from Alice (and from other resellers) before Bob sends an RD activation request to LI. In this case, however, Bob may never get \( Msg_3 \) from Alice (and from the other resellers).

To prevent Bob from waiting forever, we suggest adding a \textit{time-out} field to \( Msg_2 \). After sending \( Msg_2 \) to Alice, Bob waits till this time-out expires and then Bob may ask Alice to send \( Msg_3 \) one more time and waits for another time-out. If Bob still does not receive \( Msg_3 \) within the second time-out, he may terminate the protocol run. Note that Bob can always go back to choose \textit{Option 1} after terminating the RDS protocol execution. This means that Bob has always got a strategy to counter any misbehaviour by Alice.
CHAPTER 4. A RESELLING DEAL SIGNING (RDS) PROTOCOL

1- Alice sends Bob Msg1
   \[ \text{Msg1} = \text{RD||ASignA(RD)||RPLic} \]

2- Bob performs Bv1 & Bv2 then sends Alice Msg2
   \[ \text{Msg2} = \text{RDAsignA||ASignB(RDAsignA)||Time-Out} \]
   \[ \text{Msg3} = \{f||EPKB(ks)} \]

3- Alice sends Charlie Msg'1
   \[ \text{Msg'1} = \text{RD2||ASignA(RD2)||RPLic} \]

Protocol Verifications:
- Bv1: Checking the correctness of RPLic
- Bv2: Checking the correctness of AsignA
- Bv3: confirming that H1(ks) equals f which was provided in RPLic
- Av1: confirming that the same f is used in both AsignA and AsignB
- Av2: checking the correctness of AsignB

Alice performs Av1 & Av2. If they are positive, Alice does not send Msg3 to Bob, but signs RD2 with Charlie.

4- Charlie performs Cv1 & Cv2

5- Charlie sends LI an RD2 activation request

6- After Alice sends Charlie Msg'1, she sends Bob Msg3

7- Bob performs Bv3

Option 1: Bob sends an RD1 activation request to LI

Option 2: Bob sends an RD3 (RD signed with Sally) activation request to LI

Option 3: Before Time-out expires, Bob sends an RD activation request to LI

During time-out, Bob signs RD3 with Sally.

Signed RD3

Figure 4.18: Schematic figure of the countermeasure to the AoAF1 Attack
Specifying a time-out value for $Msg_2$: There are a number of factors that should be considered when specifying a value of the time-out. These factors are:

- $T_{Del-Msg_2}$: The time interval during which $Msg_2$ should be delivered to Alice across the network.
- $T_{A_{v_1}+A_{v_2}}$: Time required for Alice to perform verifications, $A_{v_1}$ and $A_{v_2}$.
- $T_{Prove2C}$: Time required for Alice to prove to Charlie (i.e. Prove2C) that Bob has indeed signed a deal, RD. This indicates the time taken by Charlie to perform verifications, $A_{v_1}$ and $A_{v_2}$, in order to make sure that Bob has indeed signed the RD.
- $T_{Msg_3} = T_{Gen-Msg_3} + T_{Del-Msg_3}$: Time required for Alice to generate $Msg_3$ and to deliver it to Bob.
- $T_{Msg'_1} = T_{Gen-Msg'_1} + T_{Del-Msg'_1}$: Time required for Alice to generate $Msg'_1$ (the 1st message of an RDS protocol run between Alice and Charlie) and to deliver it to Charlie.
- $T_{C_{v_1}+C_{v_2}}$: Time required for Charlie to perform verifications, $C_{v_1}$ and $C_{v_2}$, after she receives $Msg'_1$ from Alice.
- $T_{RD_2-ActReq} = T_{Gen-RD_2-ActReq} + T_{Del-RD_2-ActReq}$: Time required for Charlie to generate an $RD_2 - Act - Req$ and to deliver it to LI.

In addition to the factors above, additional delays caused by network congestions and/or delays variation due to capabilities of the signers’ devices should also be considered. Based on all these factors, the inequality (4.15) specifies a value of the time-out Bob should wait before he terminates the RDS protocol run after sending $Msg_2$ to Alice.

$Time - out < T_{Del-Msg_2} + T_{A_{v_1}+A_{v_2}} + T_{Msg'_1} + T_{C_{v_1}+C_{v_2}} + T_{RD_2-Act-Req}$ (4.15)

Attack 2 on the Abuse-freeness ($AoAF_2$): The formal verification of the abuse-free property has also led to the identification another attack on the abuse-free property (i.e. $AoAF_2$). In this attack, Bob makes use of Alice’s binding signature on another item contained in $Msg_1$ to prove to a third party that Alice...
has engaged in an RDS protocol with Bob. This engagement could motivate the third party to sign a better deal with Bob.

The following scenario illustrates how $AoAF_2$ may be committed. Suppose that $RP_{lic}$ provided in $Msg_1$, is of the form,

$$RP_{lic} = (Lic||f||Sig_{LI}(Lic||f)||Sign_A(Lic||f)),$$

where $Sign_A(Lic||f)$ is to prove that the license, $Lic$, is Alice’s license. In addition, suppose that Alice and Bob have agreed to sign a deal, $RD$, for $Lic$. To do so, Alice first signs this $RD$ and sends it along with $RP_{lic}$ to Bob in $Msg_1$. After receiving $Msg_1$, Bob could use Alice’s binding signature, $Sign_A(Lic||f)$, to prove to another reseller (e.g., Sally) that Alice has indeed started an RDS protocol run with Bob. Since Sally can verify that the signature, $Sign_A(Lic||f)$ is Alice’s signature, Sally will be assured that Alice has indeed engaged in the RDS protocol run with Bob. In this way, Bob may be able to coerce Sally to resell her license with a price less than what has been agreed with Alice. Thus, the RDS protocol is not abuse-free.

**Countermeasure to $AoAF_2$:** Our RDS protocol has taken the following measure to counter the $AoAF_2$ attack. We have designed $RP_{lic}$ with the following form,

$$RP_{lic} = (Lic||f||Sig_{LI}(Lic||f)).$$

Since this form does not contain any information linking Alice’s identity to $Lic$, it is hard for Bob to mount $AoAF_2$.

**Other Attacks: Eavesdropping Attack**

In addition to analysing attacks on the abuse-free property, the formal verification of the RDS protocol has also led to the identification of an eavesdropping attack. If the underlying communication channels do not have authentication and confidentiality protections, a dishonest reseller may be able to forge, delay, and modify messages sent between the buyer and LI. As a result, the fairness and the abuse-freeness property of the RDS protocol will be compromised. The following scenario gives more details.

To sign a deal, $RD$, with Bob, Alice sends him $Msg_1$. Bob then replies with $Msg_2$ after positive verifications of $B_{V2}$ and $B_{V3}$. Once Alice receives $Msg_2$ from Bob, she can simply refuse or delay sending $Msg_3$ to Bob. As she does not want Bob to successfully activate the $RD$ with LI, she may eavesdrop on the channel between Bob and LI. When Bob sends LI an $RD$ activation request to activate the $RD$ partially signed by Alice and Bob, Alice may intercept this request. She may change the message content such that LI rejects Bob’s request. For example, if this request contains $(A, ASign_B, RP_{lic},$ and $Payment_B)$. Alice may remove $ASign_A$ from the request and put another cryptographic token. In this way, LI
will reject the request as it does not contain Alice’s signature. This breaks the fairness property for Bob as Alice has received $A\text{Sign}_B$ in $Msg_2$ but Bob does not have $A\text{Sign}_A$ from LI.

Here is another example. Alice may intercept Bob’s RD activation request sent to LI and then delays this request till she abuses Bob’s commitment (i.e $A\text{Sign}_B$) to the signed RD. For instance, Alice may get $A\text{Sign}_B$ from Bob’s request and then use $ks$ she holds to prove to Charlie that Bob has indeed signed a deal, RD, for a license, Lic. As shown in the $AoAF_1$ attack, Alice can then coerce Charlie to sign a better deal ($RD_2$) for Lic. Charlie then sends LI an activation request to activate $RD_2$. Later, Alice may forwards Bob’s request to LI. However, as Charlie’s request has been received by LI before Bob’s request, LI will activate Charlie’s request and reject Bob’s one. Therefore, in this case, Bob’s signature ($A\text{Sign}_B$) is abused.

**Countermeasure to Eavesdropping Attack:** To thwart the attack described above, the channel between Bob and LI should be protected with confidentiality and integrity service. By using SSL (i.e. Assumption 8) in the underlying channels between each pair of the involved entities are confidentially protected and authenticated, it is hard for the above mentioned attack to succeed.

## 4.8 Protocol Performance Analysis

### 4.8.1 RDS Protocol Computational Cost

In this section, the computational cost of the RDS protocol is evaluated. As the exponentiation operations, $\exp#$, are the heaviest computational operations in the designed protocol, the evaluation will be largely performed by computing the number of the exponentiation operations used in the protocol. Table 4.2 shows the number of the exponentiation operations for each entity in two cases: when LI is not used and when LI is used.

<table>
<thead>
<tr>
<th></th>
<th>Alice Exp#</th>
<th>Bob Exp#</th>
<th>LI Exp#</th>
<th>Total Exp#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without LI’s help</td>
<td>7</td>
<td>8</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>With LI’s help</td>
<td>no $ks$</td>
<td>3</td>
<td>9</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>invalid $ks$</td>
<td>7</td>
<td>10</td>
<td>26</td>
</tr>
</tbody>
</table>
Without LI’s help: This is the case when Alice and Bob faithfully followed the designed protocol. During an execution the protocol, Alice performs 7 Exp# (1 in the setup algorithm, 2 when ambiguously signing an RD deal, 3 when verifying Bob’s ambiguous signature, and 1 when releasing ks, i.e., encrypting ks with Bob’s public key). Also, Bob performs 8 Exp# (1 in the setup algorithm, 2 when ambiguously signing the RD deal, 3 when verifying Alice’s ambiguous signature, 1 when verifying RP of Alice’s license, and 1 when verifying ks released by Alice). In this scenario, LI is not involved, so LI performs 0 Exp#.

With LI’s help: There are two cases in which LI is involved to help. Firstly, when Alice sends Msg1 to Bob but does not release ks at Msg3. To get ks, Bob generates an RD activation request, RD−Act−Req, and sends it to LI to request for ks while requesting for a license purchase. In this scenario, Alice performs 3 Exp# (1 in the setup algorithm, 2 when ambiguously signing the RD deal). Bob performs 9 Exp# (1 the setup algorithm, 2 when ambiguously signing the RD deal, 3 when verifying Alice’s ambiguous signature, 1 when verifying RP of Alice’s license, 1 when signing RD−Act−Req, and 1 when verifying the result of RD−Act−Req received from LI). In addition, to send Bob ks along with the license, LI needs to perform 9 Exp# (1 when verifying Bob’s signature on RD−Act−Req, 1 when verifying RP included in RD−Act−Req, 6 when verifying Alice’s and Bob’s ambiguous signature, and 1 when sending Bob ks and the license which require LI’s signature).

Secondly, when Alice sends Msg3 to Bob but does not include the valid ks the message (i.e., send invalid keystone). In this case, Alice performs 7 Exp# (the same exponentiation operations she does in the case of execution the RDS protocol without LI’s help). Bob performs 10 Exp# (8 Exp# like the case of executing the protocol without LI’s help and another 2 Exp# to get a valid ks from LI, i.e., 1 when signing RD−Act−Req, and 1 when verifying the result of RD−Act−Req received from LI). To send LI a valid ks, LI performs 9 Exp# (the same as LI does when Bob does not receive ks).

It should be emphasized that (1) the 9 exponentiation operations, performed by LI, will only be performed if Bob had paid LI the price agreed between him and Alice. In other words, these operations are performed as part of the reselling process of the license for which Alice and Bob have signed RD; (2) the extra Exp# performed by Bob when he receives an invalid ks or does not receive any ks, are computational costs required for Bob to not only obtain ks but also to
activate the license on his device, i.e. completing the license purchase process.

4.8.2 RDS Protocol Communication Cost

Table 4.3 and Table 4.4 demonstrate the communication overhead of the RDS protocol. This overhead is evaluated in terms of the number and the sizes of messages exchanged between the protocols entities. To evaluate message size, the following assumptions have been used:

- A random number generator of length 512-bits is used to generate the keystone, $ks$. This length is chosen to ensure security.

- Another random number of length 128-bits is used to generate a license ID, $Lic$. The length of this random number could be changed to suit the scale of licenses to be generated by LI.

- The MD5 algorithm is used to generate a keystone fix, $f$, from the keystone, $ks$. Thus the hash value $|H_1(\cdot)|$ is 128-bits long.

- The SHA-1 algorithm is used to generate other hash values in signature generations. Thus the hash value $|H_2(\cdot)|$ and any other hash value are 160-bits long.

- The primes $|p| = 1024$ bits, $|q| = 160$ bits are used to (1) generate public/private key pair for Alice and Bob during the signing process using the CS scheme, (2) create the ambiguous signatures of Alice and Bob. To get more secure key pair, other values for $p$ and $q$ could be considered, e.g. $|p| = 2048$ bits, $|q| = 512$ bits.

- The RSA algorithm is used for encryption. The key used for RSA is 1024 bits long, thus the size of the output is in multiples of 1024-bits. Since $ks$ is 521-bits long, the ciphertext $E_{PK_B}(ks)$ using RSA is 1024-bits long.

- $RP_{Lic}$ contains LI’s signature generated using a 1023-bits long RSA private key. As the content to be signed is 160-bits long (the hash value produced by SHA-1 algorithm ), then the generated signature on $RP_{Lic}$ will be of length 1024-bits.

In summary, an execution of the RDS protocol will incur a communication overhead of 12719 bits which are computed of 3 messages (total protocol messages).
Table 4.3: Items and their size in each RDS protocol message

<table>
<thead>
<tr>
<th>RDS messages</th>
<th>Item</th>
<th>Size(bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Msg_1$</td>
<td>$RD^*$</td>
<td>4472</td>
</tr>
<tr>
<td></td>
<td>$ASign_A$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$f$</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>$h_A$</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>$s_A$</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>$RP_{Lic}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$f$</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>$Lic$</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>$Sig_{Lic}$</td>
<td>1024</td>
</tr>
<tr>
<td>$Msg_2$</td>
<td>$RD_{ASign_A}$</td>
<td>4920</td>
</tr>
<tr>
<td></td>
<td>$ASign_B$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$f$</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>$h_B$</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>$s_B$</td>
<td>160</td>
</tr>
<tr>
<td>$Msg_3$</td>
<td>$f$</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>$E_{PK_B}(ks)$</td>
<td>1024</td>
</tr>
</tbody>
</table>

* The RD deal size used in this evaluation is 4472 bits long. The deal size could change based on the terms and conditions specified in it.

Table 4.4: Communication cost of each RDS protocol message

<table>
<thead>
<tr>
<th>Protocol message</th>
<th>$Msg_1$</th>
<th>$Msg_2$</th>
<th>$Msg_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication cost (in bits)</td>
<td>6199</td>
<td>5368</td>
<td>1152</td>
</tr>
</tbody>
</table>

4.8.3 Comparison with Related Work

This section compares the RDS protocol with the related work published in the literature. The signature exchange protocol (Chen’s protocol) proposed in [8] is the most related work to the RDS protocol. Table 4.5 demonstrates a comparison between the two protocols.

As can be seen from the table, the RDS protocol has got two major advantages over Chen’s protocol. Firstly, the RDS protocol provides the abuse-freeness property whereas Chen’s protocol does not. Secondly, with the RDS protocol, the keystone, $ks$, is integrity-protected regardless of channel security measures whereas with Chen’s protocol $ks$ is only protected when integrity-protected channels are used.

Both protocols achieve the properties of strong fairness, NOO, and NOR. In the case where the keystone, $ks$, is not released by the initial signer, both protocols need help from a TTP to achieve these properties.
Table 4.5: Comparison between Chen’s protocol and the RDS protocol

<table>
<thead>
<tr>
<th></th>
<th>Chen’s Protocol</th>
<th>RDS Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong fairness</td>
<td>Yes (with help from TTP)</td>
<td>Yes (with help from TTP)</td>
</tr>
<tr>
<td>Abuse-freeness</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Computational Cost</td>
<td>12 Exp#</td>
<td>15 Exp#</td>
</tr>
<tr>
<td>Number of Msgs</td>
<td>3 Msg</td>
<td>3 Msg</td>
</tr>
<tr>
<td>NOO</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>NOR</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ks integrity</td>
<td>Yes (if the communication channel is integrity protected)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The estimation of the computational costs of the compared protocols are evaluated in terms of the exponentiation operations taken in a protocol execution. This estimation indicates that Chen’s protocol is cheaper than the RDS protocol by 3 Exp#. However, the extra 3 exponentiation operations performed by the RDS protocol have allowed us to achieve the abuse-freeness and $ks$ integrity-protections.

4.8.4 Prototyping and Evaluation

This section describes the design and the implementation of the RDS protocol. Based on this implementation, we evaluate the performance of the protocol.

4.8.4.1 RDS Protocol Design

The design of the RDS protocol, as shown in Figure 4.19, consists of three participants: License Issuer (LI), reseller (Alice), and buyer (Bob). The role of each participant is as follows:

- **LI**: It issues $ks$ and $RP$ for a license, $Lic$, and sends them to Alice. LI also resolves any fairness problem that may happen during the RDS protocol execution. Issuing $ks$ and $RP$, and resolving any fairness disputes are done during a license issuing process and an RD activation process, respectively.

- **Alice**: She wants to resell her license, $Lic$, so she uses $ks$ and $RP$ to sign an RD deal with Bob. Alice also needs to generate (1) a public/private key
pair\footnote{This pair and Bob's pair (below) are created as part of the CS scheme setup algorithm and it is only used during the RD signing process.} to be used for RD signing, (2) public parameters, described in Section A.1, i.e. $p$, $q$, and $g$. She shares these parameters and her public key with Bob.

- **Bob**: He receives the public parameters from Alice. He also needs to generate his public/private key pair and share his public key with Alice. In addition, Bob needs to sign an RD deal with Alice to buy $Lic$ from Alice. Moreover, Bob could ask LI to resolve any fairness problem in case Alice does not complete the signing process fairly.

The design of the RDS protocol consists of four main parts:

- **Issuing $ks$ & $RP$**: This part is achieved prior to the execution of the RDS protocol. It is executed by LI to issue Alice $ks$ and $RP$ containing $f$. LI performs this operation while it is issuing a resalable license, $Lic$, to Alice for the first time. As the operation of issuing $ks$ & $RP$ is not supported by the current DRM system, this operation needs to be implemented to support the operation of the RDS protocol.

- **RD Pre-Signing**: This part is done by both Alice and Bob to (1) agree on and exchange the public parameters $p, q$ and $g$, (2) generate their public/private key pairs (used only for the signing process), (3) exchange their respective public keys.

- **RD Signing**: This part is run between Alice and Bob to sign a Reselling Deal (RD) for a License, $Lic$. In this part, Alice and Bob perform the actual signing process of the RD deal as described in Section A.1. In the RD signing process, Alice and Bob need to engage in the following operations: (a) generating ambiguous signatures, (b) verifying these signatures, (c) verifying binding signatures.

- **RD Resolution**: This part is executed between Bob and LI if Bob does not receive $ks$ from Alice and wants to continue to purchase $Lic$ from Alice. In this case, Bob will send LI an RD activation request containing (1) ambiguous signatures for both of him and Alice, (2) payment, (3) agreed RD, and (4) Bob’s signature on this request. On the receipt of this request, LI
needs to verify its validity. Thus, this phase involves the following operations: (a) digital signature generation, and (b) payment method, (c) digital signature verification. Note that, the payment method is assumed to be secure and similar to the one used in the current DRM systems.

4.8.4.2 RDS Protocol Implementation

Figure 4.20 describes the structure and the relation of the implemented classes which are executed during a run of the RDS protocol.

As outlined in Figure 4.20, the RDS protocol implementation consists of seven classes. There are three classes containing “main method”. They are Class Alice (Reseller), Class Bob (Buyer), and Class LI. In addition, there are four other classes, KeysGeneration, RSA signature, RSA encryption, and RSA decryption to support the main three classes. For example, prior to sending the keystone, $ks$, to Bob, Alice uses RSA encryption class to encrypt $ks$ with Bob’s public key.

The four parts of the RDS protocol, mentioned above, are implemented as follows.

**Issuing ks & RP** is achieved by having LI run its class to generate the keystone, $ks$, and the keystone fix, $f$, using the method, GenerateksAndf. LI then issues the Reselling Permission, $RP$, containing $f$ using the method, IssueRP. LI then sends $ks$ and $RP$ while it sends the resalable license, $Lic$, to Alice.
**CHAPTER 4. A RESELLING DEAL SIGNING (RDS) PROTOCOL**

**Figure 4.20:** Classes created for the implementation of the RDS protocol

**RD Pre-Singing** is done by having Alice run her class to generate the public parameters $p$, $q$, and $g$ using the method `SetupA`. Note that these parameters are of class `BigInteger` to ensure their security. Alice also generates her public/private key pair ($A_{SetupPubK}$ and $A_{SetupPrvk}$) using the method `SetupA`. Alice then sends the values $g$, $p$, $q$, and $A_{SetupPubK}$ to Bob using the method, `SendMessageA`. In this case, Alice will be acting as a client and Bob will be run as a server waiting for a connection from Alice. Once Bob receives these values, he generates his setup public/private key pair ($B_{SetupPubK}$ and $B_{SetupPrvk}$). Bob then sends his $B_{SetupPubK}$ to Alice using the method `SendMessageB`. This ends the RD Pre-Signing part.

**RD Signing** is implemented as follows. Once, Alice receives $B_{SetupPubK}$ from Bob, she uses the method, `AmbiguousSignA` to ambiguously sign an RD deal. She then uses the method `SendMessageA` to send the signed RD to Bob. Using the method `VerifySignA`, Bob will verify Alice’s ambiguous signature and
using the method \textit{VerifyRP} he will verify the authenticity of $RP_{lic}$ sent with the signed RD. Bob then uses the method \textit{AmbiguousSignB} to ambiguously sign RD and send it back to Alice using the method \textit{SendMessageB}. Using the method \textit{VerifySignB}, Alice verifies Bob's ambiguous signature and then sends $ks$ to Bob using the method \textit{ReleaseKs}. Finally, Bob will use the method \textit{VerifyKs} to verify $ks$ received from Alice.

\textbf{RD Resolution} is accomplished by having Bob send an RD activation request to LI using the method \textit{RDactivationReq}. On the reception of this request, LI will use the method \textit{VerifyRDActRequest} to verify whether this request can be activated (i.e. perform LI's verifications described in Section 4.6). LI then sends $ks$ to Bob while sending the activated license to him.

\subsection*{4.8.4.3 Hardware and Software Architecture}

To prototype the RDS protocol, the following hardware and software have been used. We have used a desktop computer running Windows XP version 2003 with a 2.66 GHs Intel Core2 and 1024 MB of RAM. The timing results of the RDS protocol execution presented here are based on this computer specification.

To implement the RDS prototype, we have used JAVA 2 Platform, Standard Edition (J2SE). JAVA is chosen because it supports a set of standard security interfaces. Examples of these interfaces include the hash functions SHA-1 and MD-5, and the symmetric key encryption algorithms DSA and AES.

\subsection*{4.8.4.4 RDS Protocol Evaluation}

The evaluation of the RDS protocol consists of two parts: performance evaluation presented in Section 4.8.4.4.1, and security test evaluation discussed in Section 4.8.4.4.2.

\subsubsection*{4.8.4.4.1 Performance Evaluation}

To evaluate the computational cost of the RDS protocol, we have run the RDS prototype and measured the time taken to execute the protocol under two scenarios. The first scenario is when only Alice and Bob are involved in the protocol execution, i.e. \textit{when LI is not used}. The second scenario is when Bob does not receive the keystore, $ks$, and invokes LI to get $ks$, i.e. \textit{when LI is used}. For each these scenarios, the RDS protocol is executed 10 times in which 10 different RDs are used, i.e. different license prices and different terms and conditions are considered. The average time refers to the
average values of the execution times of these 10 cases. The results of these two scenarios are shown in Figure 4.21.

As shown in the figure, when LI is involved, the time taken to perform the RDS protocol (4134.14 milliseconds) is approximately 27% more than the time taken when LI is not invoked (2929.07 milliseconds). This extra time is caused by two reasons. Firstly, LI needs to perform LI’s verifications described in Section 4.6. Secondly, the extra communication between Bob and LI.

To evaluate the performance of the RDS protocol when the market power is applied (i.e. signing 3 deals for one license), the RDS protocol is executed 10 times for each of the following cases: (1) fair signing: Alice and Bob faithfully follow the protocol; (2) ks not sent: Alice receives \( M \sigma_2 \) but does not send \( ks \) to Bob; and (3) invalid ks: Alice receives \( M \sigma_2 \) but send invalid \( ks \) to Bob. In the case (2) and (3), Bob invokes LI to obtain \( ks \) while activating a signed deal. The time measurements are then taken by computing the average of the 10 execution rounds for each case. The results of these cases are shown in Figures 4.22, 4.23, and 4.24.

These figures show two remarks. Firstly, as shown in Figure 4.24, LI’s performance does not affect by the number of deals signed for one license. This is because, if LI has received, for example, 3 deals for one license, LI only activates the first deal received (i.e. first-come first-served). In addition, once a deal is
4.8. PROTOCOL PERFORMANCE ANALYSIS

Figure 4.22: The average time taken by Alice vs a number of deals signed for one license

Figure 4.23: The average time taken by Bob vs a number of deals signed for one license

activated for the license, LI rejects any further deals which may be received later. Secondly, the cost of apply the market power is only imposed on the entity that may benefit from it. For example, if Bob wants to collect 3 deals from different resellers for one license, he needs to run the RDS protocol 3 times with these resellers. As shown in Figure 4.23, the time required for Bob to sign these 3 deals depends on the behaviour of the resellers (i.e. the three cases mentioned above,
fair signing, $ks$ not sent, or $ks$ invalid). The same can be applied to the case of Alice if she wants to offer her license to 3 buyers. This is shown in Figure 4.22.

4.8.4.4.2 Test against Security Attacks To evaluate the security (i.e. the integrity of a deal and keystone and the authorisation of using a reselling permission) strength of the RDS protocol, we have tested a number of identified attacks against the protocol prototype. These attacks could be mounted either by Alice or by Bob. The security evaluation has shown that the prototype can thwart all these attacks. When an attack is detected, the prototype terminates the protocol execution as designed. To further explain this evaluation, we have used a test case for each of Alice’s and Bob’s attacks.

**Alice’s Attacks:** Examples of the attacks mounted by Alice include (a) a reselling deal alteration, (b) a reselling permission swapping, and (c) an invalid keystone.

**A reselling deal alteration attack:** In this attack, Alice tampers with a reselling deal, $RD_1$, agreed during a negotiation phase. Alice, in $Msg_1$, may alter a license ID agreed in $RD_1$ to another less valuable license ID. This attempt illustrates the scenario where a reseller wants to cheat a buyer by reselling a higher-price license, agreed in the deal, with a lower-price one, sent in RD in $Msg_1$. In our prototype, before Alice sends $RD_1$ in $Msg_1$ to Bob, we deliberately had Alice change the license ID, agreed in the negotiation phase, to another license.
ID. Upon receiving $Msg_1$, the first verification, Bob performs, is to verify that the license ID negotiated is identical to the one in the deal received, i.e. Bob performs verification $B_{V1}$, depicted in Figure 4.4. With this verification, Bob can detect the license ID alteration attack and can terminate the protocol execution before signing $RD_1$, even before performing heavy verifications such as $B_{V2}$ and $B_{V3}$. Any other alteration in the deal, Bob can detect it with verification $B_{V3}$ as in this verification, Bob uses his copy of the deal to verify Alice’s ambiguous signature.

A reselling permission swapping attack: In this attack, also in $Msg_1$, Alice swaps a reselling permission for a license, agreed in a deal $RD$, with another less valuable permission. This attempt demonstrates another scenario where a reseller wants to cheat a buyer by reselling a higher-price license, agreed in the deal, with a lower-price one, sent in RD in $Msg_1$. In our prototype, prior to sending $Msg_1$ to Bob, we had Alice replace this permission with another one. Like the case of detecting the license ID alteration attack, with verification $B_{V1}$ Bob can detect the license ID contained in the received permission is not identical to the one contained in the agreed deal.

An invalid keystone attack: In this attack, in $Msg_3$, a malicious Alice sends Bob a keystone, but an invalid one, i.e. a keystone which is not corresponding to the keystone fix contained in the reselling permission received in $Msg_1$. This attack demonstrates a case where a reseller wants to have a buyer’s binding signature on a deal but the buyer can not have the reseller’s binding signature on the same deal. In our prototype, we implemented this case by making Alice send Bob an invalid keystone in $Msg_3$. Once $Msg_3$ is received, with verification $B_{V4}$, Bob can detect that the keystone is invalid. Bob is given a choice to initiate an RD activation process with LI to obtains the valid keystone along with an activated license (i.e. Alice’s license which can be accessed on Bob’s device).

Bob’s Attacks: These attacks may be committed while Bob recovering a keystone, $ks$, from LI if Alice does not send $ks$ or sends an invalid $ks$ to Bob. Examples of the attacks mounted by Bob include (a) an insufficient payment, and (b) a reselling deal alteration.

An insufficient payment attack: In this attack, Bob attempts to make a

---

4Note that at the end of a deal negotiation phase, each of Alice and Bob has an identical deal, RD, containing a license ID, agreed price for the licence and other terms and conditions.

5As the reseller maintains the keystone, $ks$, she can have the buyer’s binding signature on the deal once she receives $Msg_2$ from a buyer.
payment less than the amount agreed in a reselling deal ambiguously signed by Alice. This case illustrates a scenario where a buyer wants to cheat LI by making an insufficient payment. Note that with this attack, a reseller will not be affected as he will claim from LI a payment equal to the amount stated in the deal but not equal to the amount paid to LI. In our prototype, we implemented this case by allowing Bob to make a payment less than that is specified in the deal contained in the activation request. Upon receiving this request, LI can detect this attack while performing verification $LI_{V1}$ (see Section 4.6). LI then stops the protocol execution.

A reselling deal alteration attack: In this attack, Bob tampers with a reselling deal, $RD_1$, ambiguously signed by Alice and received in $Msg_1$. Prior to signing his ambiguous signature on $RD_1$, Bob may modify the payment stated in $RD_1$, $P_1$ into a smaller amount, $P_2$. Bob then ambiguously signs $RD_1$ and contains $RD_1$ into an RD activation request and sends it to LI. Bob also makes the payment, $P_2$, to LI. In this case, the payment verification, $LI_{V1}$, described in Section 4.6 will be positive. This attempt shows a scenario where a buyer wants to cheat a reseller by paying a little-price than agreed in the deal. In our prototype, before Bob sends an RD activation request to LI, we intentionally allowed Bob to change the payment, agreed in $RD_1$, to another little payment. When receiving the request and while performing verification $LI_{V1}$, LI will not notice this attack. However, with verification, $LI_{V5}$ (see Section 4.6), LI can detect that $RD_1$ used in Alice’s ambiguous signature is different than that is used in Bob’s ambiguous signature. LI then terminates the protocol execution.

A reselling permission swapping attack: In this attack, like the case of Alice mentioned above, Bob may swap a reselling permission received in $Msg_1$ with another high-value permission. In an RD activation request, Bob then contains this permission to get a high-value license. In our implementation, we allowed Bob to do so. However, after receiving the request and while performing verification $LI_{V3}$, described in Section 4.6, LI can detect this attack and terminate the protocol execution.

---

6Note that, unlike the case of the insufficient attack, in the deal alteration attack, LI will not be affected but Alice. This is because, LI will only forward the payment stated in the deal to Alice.
4.9 Chapter Summary

This chapter has first given a survey of the current approaches of designing a contract signing protocol. It has then showed that none of the existing protocols can be adopted to sign a Reselling Deal (RD) in the license reselling scenario. The chapter has then presented a fair and abuse-free contract signing protocol (i.e. RDS protocol). This protocol is designed such that it does not make use of a dedicated TTP to help with achieving fairness. The protocol is designed to allow a reseller and a buyer to fairly sign a contract called *(Reselling Deal)* to support fair license reselling. The protocol design has made full use of the special features provided by two building blocks: (1) the CS scheme that provides both weak fairness and ambiguity, and (2) the existing license distribution infrastructure (i.e. LI). By combining the weak fairness property of the CS scheme and the existing LI, our protocol achieves strong fairness. Furthermore, by utilising the ambiguity of the CS scheme and the joint signature of both Alice and Bob on the RD contract, the protocol provides abuse-freeness. In addition, it has also shown that our protocol has a range of potential applications, including supporting the reselling of both digital licenses and access permissions.

Informal analysis of the protocol design has been conducted. In addition, to informally analysing our RDS protocol, formal verification tools have been investigated. Through the investigation, the Mocha model checker is found to be the most suitable tool for the verification of the RDS properties. The chapter proceeded to formally verify the protocol using the Mocha model checker. The verification results have shown that the fairness and the non-repudiation properties are satisfied. On the other hand, the verification has led to the identification of two attacks on the abuse-freeness property, and we have discussed countermeasures to these attacks.

Moreover, this chapter has presented a theoretical performance analysis of the RDS protocol. A comparison between the RDS protocol and its related work is conducted, and the comparison results showed that the RDS protocol performs better. To verify the theoretical analysis, a prototype of the RDS is then implemented using JAVA. Using this implementation, further performance evaluation and security test ware conducted. The results of this evaluation confirmed the theoretical analysis results.
Chapter 5
A Fair and Secure License Reselling Protocol

This chapter introduces the design and analysis of a novel protocol suite, called Fair and Secure License Reselling (FSLRP) to support a single reselling of a digital license. In detail, Section 5.1 gives an overview of the building blocks used in FSLRP design. Section 5.2 presents the design preliminaries of FSLRP, namely notations, assumptions and requirements. Section 5.3 gives an overview of the FSLRP suite before describing it in detail in Section 5.4. Section 5.5 analyses FSLRP against potential threats and attacks while Section 5.6.2 presents a comparison of the FSLRP suite with related work. Finally, Section 5.7 concludes the chapter.

5.1 Building Blocks

This section gives an overview of the main building blocks used in the design of the novel FSLRP suite. These building blocks are market power, License Revocation List (LRL), existing license distribution infrastructure (LI and DRM client), and Reselling Permission (RP).

5.1.1 Market Power

In a real-life marketplace, market power is defined as the ability of a seller and a buyer to influence the price of a product being sold in the marketplace [133]. In this marketplace, the seller and the buyer are two separate decision-making
entities. Each has its own decision variables. These variables could influence a product price which is offered in the marketplace. On one hand, the seller wants to maximize his profit. He may do so by using one of the following approaches: (1) offering his product at a price that is as high as possible; (2) offering his product at a price lower than others’ but selling as many copies as possible. The buyer, on the other hand, would like to pay as little as possible for an item. To get a lower or the lowest price, the buyer may need to search for and compare as many offers as possible before finding a better or the best offer, i.e., the one with lower or the lowest price, thus minimising his spending. Therefore, to support the monetary interests of both the resellers and the buyers, the reselling facility should allow the reseller of a license to sell it at any price and allow a buyer to search for and be able to choose one matching his price expectation. While supporting this market power, it should not be possible for either the two entities to cheat or take any advantage over the other.

In designing the FSLRP suite, we have implemented this market power through the use of three mechanisms: (1) a Reselling Deal (RD), (2) ambiguous signatures offered by the Concurrent Signature (CS) scheme, (3) an existing license distribution infrastructure including a License Issuer (LI), and a conditional keystone released by LI. With these supports, both resellers and buyers will have the freedom to set or choose their respective deals. A reseller can sign as many deals with as many potential buyers as possible. These deals may be at different prices. Similarly, a buyer could sign as many deals as possible with different resellers. All these deals will be inactive till one of them is submitted to, and accepted, by LI. These inactive deals bear both the resellers’ and the buyers’ ambiguous signatures. Ultimately, it is up to a buyer to decide which deal he would like to select, i.e. to activate. When a buyer wants to activate a particular RD, he sends this RD along with the required payment to LI. LI will then turn the corresponding ambiguous signatures on the RD into binding signatures. This is done by releasing the keystone used in generating these signatures, thus activating the RD chosen by the buyer.

5.1.2 Existing License Distribution Infrastructure

As our reselling facility is built on the existing license distribution infrastructure, this section describes this infrastructure. The infrastructure mainly consists of two components: a License Issuer (LI) and a DRM client.
5.1. BUILDING BLOCKS

5.1.2.1 License Issuer (LI)

A License Issuer (LI) is a partner of a content owner. In a typical DRM system, LI performs the following functions: (1) getting Content Encryption/Decryption Keys (CEDK) from content owners who have encrypted their digital contents with these keys; (2) assigning usage rights to DRM-protected contents, and (3) generating and distributing digital licenses which specify usage rights controlling the access to DRM-protected contents. Typically, a digital license contains CEDK, license-ID, content-ID, and usage right specifications. LI is also responsible for authenticating DRM clients representing consumers. Of course, prior to issuing licenses to consumers (or to DRM clients), LI makes sure that the consumers have made payment for the requested license [134, 135].

5.1.2.2 DRM Client

A DRM client is a secure and tamper-proof software loaded on consumers’ devices. It enables consumers to download and access encrypted contents. The client also receives licenses from LI, authenticates both the licenses and the contents, and uses CEDK keys to decrypt the contents before sending them to the content rendering software for rendering. In addition, it is responsible for enforcing usage rights associated with the digital contents. In other words, whenever digital contents are received on consumers’ devices, DRM clients, installed on these devices, are launched at this point to extract and decrypt the content. The DRM client then consults the usage rights before passing the contents to media players (e.g., Image Viewer or Video Player) on the devices. If these usage rights have expired, the DRM clients inform the media players that the contents have become unusable [136, 138].

Each DRM client is provided with a unique private key and the corresponding public key that is certified in a digital certificate. In addition to the public key, this certificate contains information such as device ID, software version, and serial number. A certificate identifies a DRM client, and also certifies the binding between the client and the key pair. This enables an LI to securely authenticate the DRM client using the standard PKI procedure. To enable LI to authenticate a DRM client, the certificate containing the public key of the DRM client is transferred from the DRM client to LI during an execution of a license acquisition protocol. As in the case of the CEDK key, the DRM client’s private key must not leave the DRM client [134].
To achieve authentication and to establish a confidential channel between a DRM client and LI, the SSL protocol could be used. The procedure is as follows. Each of the DRM client and LI obtains a public-key certificate, digitally signed by a certificate authority. The client and LI first execute the SSL handshake protocol. With this, both entities prove to each other that they are in possession of a private key corresponding to the public key certified in their respective certificates. The two entities also agree upon a session key to encrypt further communications. These protected communications include payment transactions from the client to LI, and the CEDK key from LI to the client. The client would then use this key to decrypt the purchased content [137].

5.1.3 License Revocation List (LRL)

LRL is an up-to-date list of previously issued licenses that have been revoked. These licenses are revoked for different reasons, e.g. they may have been resold, or the private keys associated to the licenses have been compromised. In our case, when a resalable license is resold, it will be added into LRL. This indicates that this license is revoked because it has been resold.

Methods to terminate (revoke) the use of a license can be classified into two categories: hardware-based and software-based. In a hardware-based method, the client is equipped with a special hardware (trusted) device. This trusted device is designed to terminate the use of a license under certain conditions. For example, in the case of reselling a license, when a reseller has resold the license, the trusted device automatically revokes the license, thus preventing its previous owner from accessing it again. A software-based method, on the other hand, uses a mechanism called a License Revocation List (LRL) to terminate the use of a resold license. In the case of reselling a license, when a license is resold it is added to an LRL. This LRL should have been installed on a reseller’s device to prevent it from using the resold license any more.

The hardware-based and the software-based methods may be used in stand-alone or in a combined manner. The hardware-based method was used in [61, 63]. However, this method is not cost-effective as the requirement for a special trusted hardware device imposes additional costs to the underlying reselling process. The software-based method along with the hardware-based methods were used in [50]. Conrado et al use the LRL mechanism in combination with the smart card technology. As this method requires the use of a trusted hardware device, a
smart card reader, it is more expensive than the pure software-based solution.

FSLRP, proposed in this thesis, is a software-only solution. It makes use of the LRL mechanism to revoke a resold license. The LRL list is maintained by LI. As will be discussed in Section 5.4 to complete a reselling of a license (i.e. to receive a payment for his resold license), a reseller has to get an up-to-date LRL from LI. This LRL contains the ID of the license being resold. Once this LRL is received, the reseller has to install it on his device to revoke the license. Upon the revocation, the DRM client installed on the reseller’s device will send LI a message to confirm that the license has been revoked. This message assures LI that the reseller will not be able to continue using the license which has been resold.

To ensure a correct license revocation process, each DRM client, resident on a consumer’s device, must be provided with the latest copy of LRL issued by LI. In other words, whenever a consumer acquires a license from LI, the consumer should be issued with an up-to-date LRL along with the license. Whenever the consumer wishes to access any license on his device, the DRM client will first check the installed LRL to make sure that this license is not on the LRL. If the license ID is on the LRL, the DRM client will not be able to use the license. Also, each time the consumer has confirmed that he has resold a license, LI should add the ID of this license to LRL and generate an up-to-date LRL. To complete the reselling process, the consumer (reseller) should get and install this up-to-date LRL on his device. His DRM client should also notify LI by sending a confirmation message. More details to be discussed in Section 5.4.3.

5.1.3.1 LRL Types

There are two types of LRL, Global-LRL and Private-LRL. This section describes these LRL types.

A Global-LRL is a signed list which contains all the revoked licenses issued by LI. This list is embedded within each license issued by this LI. It is installed on a consumer’s device on two different occasions. The first occasion happens when a license is being issued to a given consumer’s device. LI appends an LRL to the issued license and sends both items to the consumer. While installing the license on his device, the LRL associated to the license is also installed on the device. Every time the license is used, the associated LRL is checked. If the ID of this license is on the LRL, the license cannot be used. Therefore, the associated
CHAPTER 5. RESELLING DEAL METHOD

content cannot be accessed.

The second occasion takes place when a license is being resold. As will be discussed in Section 5.4, when LI has accepted an RD activation request from a buyer, LI will generate an up-to-date LRL. This will include the ID of the license which is stated in the RD. The reseller has to obtain and install this up-to-date LRL to revoke his resold license before he claims for the payment from LI.

A license revocation approach making use of Global-LRL is efficient for LI because of the following reason. To issue a Global-LRL, LI only needs to issue, sign and manage one list for all his consumers. This makes the Global-LRL cost-effective. However, there is one problem with the use of this Global-LRL to support license transfer from a reseller’s device to a buyer’s device. Once a license is resold, the ID of this license should be added into the Global-LRL to prevent the reseller from using it again. When the buyer gets the license, he will also receive an up-to-date Global-LRL containing the ID of the resold license. As this LRL will be installed along with the license on the buyer’s device, the buyer will not be able to access this license on his device. This problem is called a license ID problem.

To overcome this problem, we suggest the following solution. Once the reselling of the license has gone through, LI re-issues this license such that the old ID is replaced with a new ID. In other words, when LI transfers this license from the reseller to the buyer, LI assigns a new ID to this license, and changes it to a non-resalable one. Thus, after the buyer installs the license and the associated LRL, he can access the license as the installed LRL does not contain the new ID of this license.

A Private-LRL is a signed list that only contains revoked licenses belonging to a particular consumer. In contrast to the Global-LRL that is used to revoke all the licenses issued by a particular LI, a Private-LRL list is used for revoking licenses issued to a particular consumer. The list is embedded within each license issued by LI to a specific consumer. This LRL is only installed on a consumer’s device from which a license is being installed/revoked. In more detail, when LI issues a license to a consumer, LI also issues the consumer a Private-LRL containing all the revoked licenses that have ever been issued to the identity of the consumer. When the consumer wishes to resell his license, it should be revoked to prevent him/her continuing to use it. In other words, once a license has been resold, i.e. when LI has received the payment and has accepted RD
submitted by the buyer, LI will add the ID of this license onto the consumer’s
private-LRL. LI then sends this private-LRL to the consumer who must install it
on his device before the DRM client sends LI a confirmation message. Without
this confirmation message, the consumer cannot claim for the payment from this
reselling. By installing the private-LRL on his device, the consumer will no longer
be able to access his license. It is worth noting that, if the Private-LRL method
is used, the license ID problem discussed above can be avoided.

A Private-LRL could be implemented using the consumer’s identity. e.g.
consumer’s public key. When LI is about to issue a license to a consumer for
the first time, LI creates a Private-LRL with empty entry in the identity of the
consumer, i.e. his public key. LI then embeds the Private-LRL into the license
being issued to the consumer. This LRL is kept empty till one of the consumer’s
licenses is to be revoked. This could happen, as in our case, when LI is informed
that the consumer (the reseller) wants to resell his license (i.e. when LI has
received and accepted an RD from a buyer). In this case, LI will add the ID of
this license onto the reseller’s Private-LRL to update it. After this addition, LI
then sends this updated Private-LRL to the consumer who has to install it on his
device should he want to complete the reselling process. Once the Private-LRL
has been installed, this consumer can no longer access his license.

5.1.3.2 Delivering LRL to Reseller

As discussed above, a resold license should be revoked from the reseller’s device
to prevent the reseller from continuing to use it. To revoke this license, an up-
to-date LRL containing the ID of the resold license should be installed on this
device. To do so, this LRL should first be delivered to the device. This LRL
delivery could be realised by one of two modes: pull mode and push mode.

5.1.3.2.1 Pull Mode A pull mode is an LRL delivering method by which
a reseller fetches an LRL from LI’s website before installing it on his device in
order to revoke a license. Once LI has accepted a given RD for a license, e.g.
Alice’s license, LI can upload an up-to-date LRL into a read-only-directory. This
up-to-date LRL contains the license’s ID. LI will post in this directory too stating
that Alice’s license has been resold. It is the responsibility of Alice to regularly
check this directory to confirm whether her license has already been resold. If
it has, Alice should download the up-to-date LRL on her device. Alice can then
install it to revoke her license as discussed in Section 5.4.3.

5.1.3.2.2 Push Mode  With the push mode, an up-to-date LRL is sent by LI to a reseller. It works as follows. Once LI has validated an RD activation request, and has activated the resold license on the buyer’s device, LI will send the reseller an up-to-date LRL containing the ID of the resold license. The reseller has to install it on his device before a confirmation message is sent back to LI. Without this message, the reseller cannot claim for the payment (for more details see Section 5.4.3).

It is worth noting that the security of the license revocation software-based approach lies in two requirements. The first is that the DRM client must be secure or tamper resistant. This requirement has been addressed by the existing license distribution infrastructure (see Section 5.1.2.2). The second requirement is that without a valid License Revocation Confirmation (LRC) message from the DRM client, the reseller will not be able to claim the payment from LI for this license reselling. To send this LRC message to LI, the DRM client must have revoked the license on the reseller’s device. This requirement has also been addressed by the existing DRM technology as, in the current license distribution infrastructure, a DRM client must digitally sign and send LI an LRC message. Therefore, to forge such a confirmation message, one has to break the security of the DRM client. In other words, we build this license reselling facility on the capabilities of the existing license distribution infrastructure. Thus, for the license revocation part, the security level afforded by our reselling facility is not lower than that of the existing infrastructure.

5.1.3.3 The Need for Imposing a Reselling Deal Validity Deadline

Both the push and pull mode of delivering an up-to-date LRL to a reseller suffer from a similar problem. In the pull mode, LI does not have any evidence to prove that a reseller, Alice, has indeed been notified to download and install the up-to-date LRL on her device. As a result, Alice could intentionally ignore the responsibility of downloading the up-to-date LRL from LI’s website. She may falsely claim that she has not been instructed to download this up-to-date LRL, rejecting the license revocation, leaving LI in a disadvantageous position.

Similarly, in the push mode LI may not be able to get any evidence to prove that Alice has indeed received the up-to-date LRL. Alice could falsely deny that
she has received this up-to-date LRL. Consequently, Alice may continue to use the license after its reselling leaving LI in a disadvantaged position. The following scenario further illustrates how LI may be disadvantaged in these cases.

Suppose LI is offering a newly released non-resalable license at the price of £10. Alice is offering the same license at only £5. To buy Alice’s license, Bob, a buyer, has made an RD with Alice to pay £5 for the license. After Bob has paid the £5 to LI and has submitted an RD signed by him and Alice, Bob will get a fresh non-resalable license from LI with £5. If the pull mode is used to deliver the up-to-date LRL to Alice, Alice may simply ignore the responsibility of downloading an up-to-date LRL (to revoke her license). If the push mode is used, Alice may falsely claim she has not received any up-to-date LRL (even though she has) and continues to use the license. At the same time, Bob will also use the same license, i.e. Alice’s license. In these scenarios, though Alice will not be able to claim for the payment from LI for the license reselling, she will still be able to continue to use her license. In addition, Bob has gained the right to use the same license too, thus violating the content owner’s rights. In these cases, it is difficult for LI to take any further action against Alice, e.g. imposing a charge on Alice to compensate LI for the difference between the two prices (the prices of the newly released license and the second-hand one). This is because LI does not have any evidence to prove that Alice has indeed got an up-to-date LRL.

To counter the above mentioned problem, we propose to add a deadline into the signed RD (i.e. $RD_{DL}$). Before this deadline, Bob must (1) pay LI a payment stated in the RD, and (2) send LI an activation request for the signed RD. If Bob misses the deadline, he will not be able to use the signed RD to claim for the license. Similarly, before $RD_{DL}$, Alice must install an up-to-date LRL to revoke her resold license stated in RD. Although LI cannot prevent Alice from committing the cheating, as discussed above, LI can now use the signed RD with $RD_{DL}$ to bring a charge against Alice. It is also worth noting that Alice can only commit such cheating once per LI. This is because LI can maintain a blacklist and if Alice is on this list, it would be difficult for her to resell or acquire licenses from this LI in the future.

This deadline is further divided into two sub-deadlines: Bob’s deadline, $RD_{DL_B}$, and Alice’s deadline, $RD_{DL_A}$. Before $RD_{DL_B}$, Bob should (1) make payment to LI, and (2) send an RD activation request to LI. Before $RD_{DL_A}$, Alice has to regularly check LI’s website to determine whether her license has been resold. If
her license has been resold, Alice has to install and allow her DRM client to send an LRC message to LI before the expiry of $RD_{DLA}$. This RD deadline and its terms can be negotiated by Alice and Bob and written into the negotiated RD during the RD signing phase.

The pull and push modes may be used in a hybrid manner to deliver an up-to-date LRL to Alice. For example, once Alice has seen on LI’s website that her license has been resold, she can download an update of LRL (i.e. using the Pull mode). If LI does not receive a LRC response from Alice and $RD_{DLA}$ is still valid, LI could send an LRC request to Alice to push Alice to send the LRC response. If LI does not receive a LRC response before $RD_{DLA}$, the up-to-date LRL installation should be considered as failed and Alice will not be able to claim for the resale payment. In addition, LI may use the signed RD containing the RD deadline, $RD_{DLA}$, to bring a charge against Alice. In other words, the signed RD with RD deadline serves as evidence that Alice is aware of her responsibility and the time frame to fulfil her responsibility in a reselling process.

5.1.4 Reselling Permission (RP)

This section gives a detailed description of a Reselling Permission (RP) which is a main component of the RD method. It explains what RP is and when and how RP is generated.

A Reselling Permission (RP) is a digital token authorising a license to be resold. In other words, when a license is accompanied by RP, it is considered as a resalable license. As illustrated in Figure 5.1, RP consists of a license ID, $Lic$, a reselling validity period\footnote{This validity period is an optional field in RP. An LI, for example, can use it to prevent a license from reselling within the first two years of its new release.} $RP_{period}$, within which RP is valid, a keystone fix, $f$, to be used in signing a deal, RD, during the process of reselling $Lic$, and LI’s signature, $Sig_{LI}$, to protect the integrity of RP.

RP is generated when a resalable license is generated. When LI is requested by a consumer to issue a resalable license, $Lic$, LI will create RP for this $Lic$, i.e. $RP_{Lic}$. LI then attaches this $RP_{Lic}$ to $Lic$ and sends them to the consumer.

The generation process of a permission, RP can be described as follows:

1. LI generates a random number called keystone, $ks$.

2. LI applies a hash function to this $ks$ to generate a keystone fix, $f$. 
3. LI generates an identity for the license being issued, i.e. Lic.

4. LI defines a reselling validity period for RP, i.e. $RP_{period}$. It contains two dates: Start date and End date. Within this period, RP is valid to authorise Lic to be resold. In other words, this period allows content owners to specify when a resalable license can be resold. With the reselling validity period, the content owners can have different types of resalable licenses. They can have resalable licenses which can be resold after one, two or three years after their release. Therefore, they can protect their revenue by only allow a newly released license to be resold after one or two years from its release.

5. LI then concatenates the items, Lic, f, and $RP_{period}$ and signs this concatenation, i.e. generates $\text{Sig}_{LI}(\text{Lic}||f||RP_{period})$ to protect the integrity of RP. This concatenation and signature forms $RP_{Lic}$.

### Figure 5.1: The RP structure of a License Lic

<table>
<thead>
<tr>
<th>License Identity</th>
<th>Keystone Fix</th>
<th>RP Validity Period</th>
<th>LI’s Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lic</td>
<td>f</td>
<td>StDate: EnDate:</td>
<td>$\text{Sig}_{LI}$</td>
</tr>
</tbody>
</table>

5.2 Preliminaries

This section details the notations, assumptions, and requirements used in the design of the FSLRP suite.

5.2.1 Notations

The following notations are used in the description of the FSLRP suite.

A: Alice (a reseller);

B: Bob (a buyer);

LI: License Issuer;
CHAPTER 5. RESELLING DEAL METHOD

\textit{Lic}: The identity of the license to be resold, i.e. Alice’s license;

\textit{Lic – File}: The license file containing usage rights signed and granted by LI.
\textit{Lic\textsubscript{Activated}}: A second-hand license activated by LI and can be accessed on a buyer’s device.

\textit{RD}: A Reselling Deal which includes: (a) terms and conditions for this deal, (b) the price to be paid by the buyer, (c) the RD validity period, and (d) both Alice’s and Bob’s signatures.

\textit{RD\textsubscript{DL}A}: It is a deadline for Alice to perform two tasks. Firstly, Alice regularly checks LI’s website to find out whether her license has been resold. Secondly, if Alice’s license has been resold, Alice downloads and installs an up-to-date LRL on her device to revoke her license, thus her DRM client could send LRC message to LI. In other words, LI should receive the LRC message before the expiry of the deadline, \textit{RD\textsubscript{DL}A}. Note that there is another way by which Alice can be notified that her license has been resold. In this way, once her license has been resold, LI sends Alice a message stating so and containing an up-to-date LRL to be installed on Alice’s device to revoke her license. The problem with this way is that LI can not be assured that Alice has indeed received this message. This could lead to a cheating scenario as described in Section \ref{cheating}.

\textit{RD\textsubscript{DL}B}: It is a deadline for Bob to additionally perform two tasks. In the first task, Bob makes the payment stated in the signed RD to LI. In the second task, Bob submits an RD activation request to LI to activate the signed RD. In other words, Bob must activate the RD before the expiry of the deadline, \textit{RD\textsubscript{DL}B}.

\textit{ks}: a keystone that is used to bind ambiguous signatures to their respective signers.

\textit{f}: a keystone fix that is a hash value of the keystone \textit{ks}. \textit{f} is used in creating ambiguous signatures.

\textit{PK\textsubscript{i}}: public key of entity \textit{i}.

\textit{SK\textsubscript{i}}: private key of entity \textit{i}.

\textit{M||N}: concatenation of two messages, \textit{M} and \textit{N}.
5.2. PRELIMINARIES

\( E_{PK_i} \): an asymmetric encryption using a public key of entity, \( i, PK_i \).

\( E_K \): a symmetric encryption using a secret key, \( K \).

\( \text{sig}_i \): a digital signature created by entity, \( i \), using its private key, \( SK_i \).

\( A\text{Sign}_i \): an ambiguous signature created by an entity, \( i \), using a keystone fix, \( f \) and public keys of both the entity, \( i \), and a second entity.

\( H() \): a cryptographic hash function.

\( LRC \): a License Revocation Confirmation message which is created and sent by Alice’s DRM client to LI to confirm that the license, \( Lic \), has indeed been revoked from Alice’s device.

\( LRL \): a License Revocation List which is a signed list containing all the revoked licenses issued by a particular LI.

\( T_A \): A time read from the clock of Alice’s DRM client.

\( RP_{Lic} \): Reselling Permission of the license, \( Lic \). It contains: (a) a keystone fix, \( f \), and (b) a reselling validity period for this permission, i.e. \( RP_{\text{period}} \), and (c) LI’s signature.

\( RP_{\text{period}} \): A period within which a reselling permission, \( RP_{Lic} \), is valid to be used in reselling the associated license, \( Lic \).

\( N_i \): A nonce which is created by entity, \( i \).

\( RD_{\text{Pre-official}} \): This is a pre-binding RD carrying both Alice’s and Bob’s ambiguous signatures.

\( Install-Status \): The status of installing an up-to-date LRL on Alice’s device.

\( Payment_B \): The payment which Alice and Bob have agreed on during the negotiation phase. This is the amount that Bob should pay to LI to obtain \( Lic \).
5.2.2 Design Assumptions

The following assumptions are used in the design of the FSLRP suite.

1. The payment systems used for a license buyer to make a payment to LI and for a license reseller to receive a payment from LI are secure.

2. License Issuer (LI) is a trustworthy entity. It issues fresh licenses (resalable and non-resalable) to consumers. LI additionally assists to facilitate the reselling of resalable licenses.

3. A resalable license is initially issued by LI. Each resalable license is attached with a Reselling Permission (RP) that contains a unique keystone fix and reselling validity period.

4. A Reselling Deal (RD) contains the terms and conditions of a reselling agreement negotiated between a reseller and a buyer. The RD also includes the price of the license to be resold and the two deadlines, \( RD_{D_{LA}} \) and \( RD_{D_{LB}} \).

5. Alice and Bob do not trust each other.

6. The keystone, \( ks \), corresponding to the keystone fix, \( f \), can only be used once, i.e. each license is issued with a unique \( ks \). In this way, each license can be resold just once.

7. The hash functions used in the protocol are collision-free hash functions.

8. Each entity, \( i \), where \( i \in \{ A, B, LI \} \) has a public/private key pair. The public key, denoted as \( PK_i \) (\( i \in \{ A, B, LI \} \)), has been certified by a certification authority, \( CA \). The private keys, denoted as \( SK_i \) (\( i \in \{ A, B, LI \} \)), are kept secret by their respective holders.

9. Communication channels between each pair of entities are resilient. This means that messages sent over these channels may be delayed, but will eventually arrive at their intended destination.

10. Communication channels are authenticated, integrity-protected, and confidential. These channels can be established using the Secure Socket Layer (SSL)\[95\]/Transport Layer Security (TLS)\[112\] protocol. For example, two entities can first use SSL to mutually authenticate each other and establish
a secret session key. They then execute our protocol using the secure channel established, encrypting all protocol messages with the agreed session key.

11. There is a DRM client resident on each consumer’s device. The DRM client is assumed to be tamper-proof and it is embedded with a secure timer, called a DRM Timer. This means that consumers cannot change or re-adjust this timer. The DRM client is capable of performing the following functions: (1) license decryption and usage rights enforcement, (2) license revocation, (3) LRC message creation and time-stamping, (4) signature generation (signing the LRC message), and (5) message transmission to LI (i.e. be able to send LI a time-stamped and signed LRC message).

12. The DRM Timer embedded inside each DRM client measures the time in the Coordinated Universal Time (UTC) time scale and it is synchronised with LI’s timer whenever the DRM client has contact with LI.

5.2.3 Design Requirements

The FSLRP suite is designed to satisfy the following requirements.

(R1) Support license transfer: The reselling facility should be able to facilitate the transfer of a resold license from a reseller’s device to a buyer’s device without compromising the content owner’s right. In other words, after a successful execution of a reselling process, the reseller should not be able to continue to use the license, while the buyer should be allowed to access the license on his device.

(R2) Support market power: The reselling facility should be able to: (1) allow a reseller to maximise his profit, i.e. to resell his license at the highest possible price; and (2) allow a buyer to minimise his cost, i.e. to purchase a license at the lowest possible price.

(R3) Protect content owners’ rights: The reselling facility should not weaken the security level of the underlying license distribution infrastructure. In other words, the reselling facility should not introduce additional flaws that would put content owners’ rights at greater risk. LI must be able to detect any unauthorised or illegitimate license resellings. These include multiple
resellings of a resalable license, and reselling of a non-resalable license. LI must also be able to prevent a reseller from the continued use of a resold license. In addition, LI must be able to trace any buyer if this buyer has violated any of the usage rights of a resold license.

(R4) Fair reselling: Upon the successful execution of a reselling process, (1) either both the reseller and the buyer are concurrently bound to a given RD or neither binds, i.e. neither entity receives anything useful from the signature signing process performed between them, and (2) the reseller should be able to receive the payment, specified in the RD, from the buyer if and only if the buyer has received the license from the reseller and vice versa.

R(5) Non-repudiation: Once LI has activated a given RD, the reseller should not be able to falsely deny having signed the deal, RD, with the buyer. This allows LI to add the reseller to a blacklist or impose a charge on the reseller if he refuses to revoke his license after he has signed an RD. Similarly, the buyer must not be able to falsely deny having signed the deal, RD, with the reseller. This prevents the buyer from paying less than what has been agreed on in the RD.

(R6) License legitimacy check: Anyone, e.g. a buyer or LI, should be able to verify whether a license being resold is legitimate. In other words, any entity should be able to verify whether (1) it is resalable, (2) it is still within its reselling validity period, and (3) it has not yet been resold.

(R7) Abuse-freeness: The proposed reselling facility should possess an abuse-freeness property. This means that, during the execution of an RD signing process, neither the reseller nor the buyer could get anything that can be used to prove to an outside entity that either is in control of the outcome of the RD signing process.

The above requirements can be further interpreted into the following requirements for the three entities, respectively.

For Alice (the reseller):

(A1) There should be a mechanism to allow Alice to resell her digital license at the highest possible price to maximise her profit.
5.2. PRELIMINARIES

(A2) Once a deal has gone through (i.e., after Alice has revoked the license), Alice should be able to receive the payment for the resold license.

(A3) Alice should not have to experience a situation where a deal is bound unilaterally. In other words, any dispute such as where Alice believes that a deal has been reached, but Bob disagrees or repudiates, should be easily resolvable by a third party.

For Bob (the buyer):

(B1) There should be a mechanism to allow Bob to purchase a digital license from Alice, or anybody else who has got the same license to sell, at the lowest possible price to minimise his expenditure.

(B2) Once a deal has gone through (i.e., after Bob has made the payment), Bob should be able to receive the license he has paid for.

(B3) Bob should not have to experience a situation where a deal is bound unilaterally. In other words, any dispute such as where Bob thinks that a deal has been agreed, but Alice disagrees or repudiates, should be easy to resolve by a third party.

(B4) A deal should only be binding if the payment has been made by Bob. In other words, prior to making the payment, Bob should not be held accountable for any deals he has agreed to and signed with any reseller. This requirement allows Bob to collect a number of deals before choosing that which is best for him.

(B5) Bob should be able to verify whether the license he is about to buy is legitimate.

(B6) After making payment and receiving the license he has paid for, Bob should be able to access this license on his device.

For LI (the License Issuer):

(L1) A resalable license should not be allowed to be resold an unauthorized number of times.
(L2) LI must be able to check whether a license is legitimate. In other words, LI should be able to confirm whether (1) it is resalable, (2) it is still within its reselling validity period, and (3) it has not yet been resold. With these checks, LI is able to detect and prevent reselling a non-resalable license and an authorised reselling for a resalable license.

(L3) LI should only activate a license on Bob’s device if and only if Bob has made the due payment for the license and provided a valid RD which is signed by both Alice and Bob.

(L4) LI should only send Alice the payment from the license reselling after LI has confirmed that Alice’s resold license has indeed been revoked on Alice’s device.

(L5) Under the current DRM systems, LI is able to trace a consumer (a reseller in our case) if the consumer has violated any of the license usage rights. This capability should also be supported in our license reselling facility. In other words, LI should be able to trace a buyer of a second-hand resalable license if the buyer has violated any of the usage rights of the license he has bought from the reseller.

5.3 Fair and Secure License Reselling Protocol (FSLRP) Suite: An Overview

The Fair and Secure License Reselling Protocol (FSLRP) consists of three protocols: (1) a 2-Message RD Signing (2M-RDS) Protocol, (2) a RD Activation (RDA) Protocol, and (3) a RD Completion (RDC) Protocol. The 2M-RDS protocol is executed between a reseller and a buyer. The RDA protocol is operated between LI and the DRM client resident on the buyer’s device whereas the RDC protocol is executed between LI and the DRM client resident on the reseller’s device. As illustrated in Figure 5.2, the execution of this suite is performed in three phases. In the first phase, the 2M-RDS protocol is used for Alice (a reseller) and Bob (a buyer) to sign a negotiated Reselling Deal (RD). The signatures generated in this phase are ambiguous signatures. They are not bound to their respective signers till they are activated by LI in phase 2. In the second phase, the ambiguously signed RD will be used by Bob to activate the license stated in RD, i.e. Lic, on
5.3. \textit{FAIR AND SECURE LICENSE RESELLING PROTOCOL (FSLRP) SUITE: AN OVERVIEW}

1. Alice generates her ASignA on RD and sends (ASignA||RD) along with RPLic to Bob. 
2. Bob generates his ASignB on (ASignA||RD) and then sends it to Alice. 

On the completion of 2M-RDS, both Bob and Alice are still free to sign other RDs with other resellers and buyers, respectively. 

3. Bob sends LI an activation request for the signed RD along with the agreed payment. 
4. LI performs the verifications LIV1 to LIV6. If they are all positive, LI does step 5 and step 6. 
5. LI sends Bob Lic encrypted with Bob’s public key. 

6- After sending Lic to Bob, LI performs the following:
   6.a- Mark Lic as resold 
   6.b- Create an up-to-date LRL containing Lic’s ID 
   6.c- Publish this LRL-update on his website 
   6.d- Post on his website that Lic has been resold 

7. Alice fetches the up-to-date LRL from LI’s website. 
8. Alice installs the up-to-date LRL on her device. 
9. Alice’s DRM client generates and sends LI an LRC message. 
10. LI performs the verifications LIV7-to-LIV9. 
11. LI sends Alice the payment received from Bob at phase 2.

Figure 5.2: A schematic diagram for the FSLRP suite

his device. In other words, Bob will execute the RDA protocol with LI to activate the signed RD. This is done provided that Bob (a) has paid the agreed price to LI; and (b) has sent LI the signed RD produced from the execution of the 2M-RDS protocol in phase 1. In the third phase, the reselling process will be finalised. During this phase, Alice will (1) download an up-to-date LRL from LI’s website and install this up-to-date LRL on her device; (2) execute the RDC protocol with LI to (a) enable Alice’s DRM client to send LI an LRC message assuring LI that the license being resold has been revoked on Alice’s device; and (b) collect the payment\footnote{This payment was deposited to LI by Bob in phase 2.} for this reselling. In other words, the RDC protocol assures LI that Alice has indeed revoked her resold license on her device and enables Alice to get...
the payment collected from the license reselling. Upon the successful execution of the FSLRP protocol suite, Alice and Bob should have received the payment and the activated license, respectively, thus achieving fair license reselling.

5.4 The FSLRP Protocol Suite in Detail

This section describes in detail the three protocols comprising the FSLRP suite.

5.4.1 2-Messages RD Signing (2M-RDS) Protocol

The 2M-RDS protocol is the first protocol of the FSLRP suite. This is a variant of the RDS protocol described in chapter 4. With the 2M-RDS protocol, both Alice and Bob place their respective ambiguous signature on a negotiated RD. This is to (1) achieve non-repudiation and abuse-freeness properties, (2) support market power, and (3) help Bob to check whether the license he is about to buy is resalable. Bob, in phase 2, activates these ambiguous signatures to binding ones by LI. An overview of this protocol is given in (Section 5.4.1.1) and a detailed description is given in (Section A.2).

5.4.1.1 2M-RDS Protocol Overview

As shown in Figure 5.3, the 2M-RDS protocol consists of two messages: $Msg_{1\text{RDS}}$ and $Msg_{2\text{RDS}}$. Alice first uses $f$ provided in $RP_{\text{Lic}}$ to create her ambiguous signature, $ASign_{A}$, on RD, where $ASign_{A} = ASign_{A}(RD))$. Upon creating $ASign_{A}$, Alice sends it along with $RP_{\text{lic}}$ and $\text{Lic} - \text{File}$ to Bob as $Msg_{1\text{RDS}}$. Once Bob has received $Msg_{1\text{RDS}}$, he performs the verifications $B_{V1}, B_{V2}, B_{V3},$ and $B_{V4}$. Bob then ambiguously signs RD using the same $f$ which has been used in creating $ASign_{A}$. The resulting ambiguous signature is $ASign_{B}$, where $ASign_{B} = ASign_{B}(RD||ASign_{A})$. Bob then, in $Msg_{2\text{RDS}}$, sends $ASign_{B}$ to Alice. Upon receipt of $ASign_{B}$, Alice performs the verifications $A_{V1}$ and $A_{V2}$ to confirm that $ASign_{B}$ is created by Alice.

5.4.1.2 2M-RDS protocol Analysis

The analysis of the 2M-RDS protocol is the same as that of the RDS protocol presented and analysed in chapter 4.
5.4. THE FSLRP PROTOCOL SUITE IN DETAIL

5.4.2 RD Activation (RDA) Protocol

The RDA protocol is the second protocol of the FSLRP suite. It is designed for a buyer (Bob) to activate a deal, which has been ambiguously signed by both Bob and a reseller (Alice) using the 2M-RDS protocol. Bob can initiate the RDA protocol with LI to activate the deal. Upon successful activation, Bob will receive the license, Lic, stated in the deal. The following sections give an overview of the protocol and a detailed description of the RDA protocol is presented in Section A.3.

5.4.2.1 RDA Protocol Overview

The RDA protocol, as shown in Figure 5.4, consists of two messages: \(Msg1_{RDA}\) and \(Msg2_{RDA}\). In \(Msg1_{RDA}\), Bob sends LI an RD activation request. This request contains three items: (1) \(RD_{Pre-official}\), (2) \(RP_{Lic}\), and (3) \(Payment_B\). Upon the receipt of this RD activation request, LI performs the verifications, \(LV_1\) through to \(LV_6\) described in the next section. If these verifications are all positive, LI, in \(Msg2_{RDA}\), sends Bob the activate \(Lic_{Activated}\) such that Bob can use it on his device. LI also performs two further tasks, (a) publishes on his website that the license, Lic, has been resold, so it cannot be resold again; (b) posts on LL’s website an up-to-date LRL containing the resold license ID, Lic, so Alice can download it to revoke Lic from her device (see Section 5.4.3).
CHAPTER 5. RESELLING DEAL METHOD

5.4.2.2 RDA Protocol Analysis

This section analyses the RDA protocol against the requirements set in Section 5.2.3. In detail, it analyses the protocol against the requirements, L1, L2, L3, and L5 for LI and against B2 and B6 for Bob. In this analysis, as LI is a trustworthy entity, LI will not misbehave. In other words, LI will always follow the protocol specifications and perform the protocol executions faithfully. In addition, as Alice is not involved in the RDA protocol, she will not be considered in the analysis below. This analysis will focus on any possible cheatings that may be committed by Bob.

In the following discussion, it is assumed that (1) the RD activation request contains the required payment that is (a) equal to the amount stated in the RD, and (b) equal to the amount made to LI, and (2) the request is sent before the deadline, $RD_{DLB}$. These assumptions allow us to focus our discussion here on more subtle cheating than sending an incorrect payment, or refusing to send any
5.4. THE FSLRP PROTOCOL SUITE IN DETAIL

payment at all, or sending the request after the deadline $RD_{DLB}$. As, based on $LI_{V1}$ and $LI_{V2}$, these misbehaviours will lead to rejection of the request.

**Analysis against LI’s requirements**

L1: (Preventing double reselling) With the RDA protocol, any double reselling of a resalable license can be detected using the verification, $LI_{V1.1}$. This verification allows LI to verify whether the license, $Lic$, has already been resold. If it has been resold, LI will terminate the license reselling process, thus preventing double reselling of the license, i.e. satisfying requirement L1.

L2: (Checking license legitimacy) With the verifications, $LI_{V2.2}$, $LI_{V2.3}$ and $LI_{V2.4}$, LI can verify whether the license, $Lic$, is legitimate for resale. $LI_{V2.2}$ confirms $Lic$ is still valid to be resold while $LI_{V2.3}$ enables LI to check whether $RP_{Lic}$ has indeed been issued by LI. In addition, $LI_{V2.4}$ allows LI to confirm that the license ID in $RP_{Lic}$ is identical to that in $Lic$. LI will only activate the license, $Lic$, on Bob’s device if all these verifications are positive, thus addressing L2.

L3: (Receiving Payment) The verification, $LI_{V1}$, enables LI to only proceed in the reselling process if Bob has made the payment stated in a deal, RD. This satisfies requirement L3.

L5: (Tracing buyer’s identity) As discussed in Section A.3 during the license re-issuance process, LI uses the public key of Bob’s DRM client to encrypt the license, thus binding the activated license, $Lic_{Activated}$, to Bob’s identity. This public key has been certified by LI, so LI can use this key to trace Bob in the event of Bob violating any of the usage rights specified in the activated license, thus satisfying (L5).

**Analysis against buyer’s requirements**

B2: (Receiving the license) Upon the successful execution of the RDA protocol, Bob will receive the license he has paid for, thus satisfying B2. Since LI is trusted, LI will send Bob the license if verifications, $LI_{V1}$ through to $LI_{V6}$, are all positive.

B6: License transfer the license re-issuance process, discussed in Section A.3 enables LI to bind the license, $Lic_{Activated}$, to Bob’s DRM client. Therefore, once Bob obtains this license, he can access it on his device (i.e. solve the DRM license transfer problem).

Bob’s Misbehaviour

Upon the execution of the 2M-RDS protocol, Bob will have an RD with two signatures, $(RD||ASign_A)$ and $(RD||ASign_A||ASign_B)$. $(RD||ASign_A)$ is the
deal, RD, that has only been ambiguously signed by Alice. It is sent to Bob in \(Msg1_{RDS}\) (see Section 5.4.1). \((RD||ASign_A||ASign_B)\) is the deal, RD, that has been ambiguously signed by both Alice and Bob. It is generated by Bob and sent to Alice in \(Msg2_{RDS}\) (see Section 5.4.1). In sending an RD activation request, Bob may try to send LI an RD activation request of the following form, \(\{(RD||ASign_A)||RP_Lic||Payment_B\}\), i.e. he may exclude his signature from the request. If this happens, Bob will gain nothing. This is because LI, based on \(LI_V^5\), will reject the activation request.

5.4.3 RD Completion (RDC) Protocol

The RDC protocol is the third and also the final protocol in the FSLRP suite. It is used to finalize a license reselling process. It accomplishes two tasks (1) revoking Alice’s resold license, lic (2) and delivering Bob’s payment, \(Payment_B\), to Alice. In section 5.4.3.1 an overview of the protocol is given, and in Section A.4, a detailed description of it is presented.

5.4.3.1 RDC Protocol Overview

An overview of the RDC protocol is illustrated in Figure 5.6. From the figure, it can be seen that the RDC protocol consists of two messages: \(Msg1_{RDC}\), and \(Msg2_{RDC}\). In \(Msg1_{RDC}\), after installing an up-to-date LRL obtained from LI on her device, Alice allows her DRM client to send LI a License Revocation Confirmation (LRC) message to confirm that the LRL-update has been successfully installed on Alice’s device. Upon receipt of LRC, LI will perform the verifications, \(LI_V^7\), \(LI_V^8\), and \(LI_V^9\). If all these verifications are positive, LI, in \(Msg2_{RDC}\), sends Alice the payment, \(Payment_B^3\). Upon successful execution of the RDC protocol, a fair license reselling has been accomplished. In other words, the RDC protocol is used to ensure fairness for Alice in this license reselling process. Alice will get the payment for the license she has resold, or she retains the license but without payment.

\(^3\) Note that this payment was deposited to LI by Bob during the execution of the RDA protocol.
5.4. THE FSLRP PROTOCOL SUITE IN DETAIL

5.4.3.2 RDC Protocol Analysis

The RDC protocol aims to (1) assure LI of the revocation of the resold license and (2) send Alice the payment. This aim can be interpreted into requirements A2 and L4 specified in Section 5.2.3. This section analyses the RDC protocol against these requirements, and demonstrates that Alice can gain nothing if she misbehaves with LI during the execution of a reselling process.

Alice’s Misbehaviour: As discussed in Section A.4, \( \text{Msg}^{1}_{RDC} \) is created by Alice’s DRM client which is a tamper-proof software (see assumption 11). Hence, it would be very difficult for Alice to create this message without installing the latest LRL-update on her device. In addition, once this LRL-update is installed, the client will dispatch a revocation confirmation message. Only with this confirmation, Alice could receive the payment, \( \text{Payment}_B \), from LI. However, Alice may refuse to install the LRL-update obtained from LI on her device to revoke her resold license, \( \text{Lic} \). If this happens, Alice’s DRM client will not generate \( \text{Msg}^{1}_{RDC} \) and LI will not be able to confirm this revocation. Thus, LI will not pay Alice for reselling her license. Another price for Alice to pay by committing
this misbehaviour is that she will not be able to resell her license, Lic, again. This is because, during the RDA protocol, LI has already marked this license as resold. In other words, what Alice can gain from this misbehaviour is that she may continue to use her resold license, but she will not receive any payment for this unsuccessful reselling and will not be able to resell this license again (i.e. she has effectively converted her resalable license to non-resalable one).

For the requirements A2 and L4: Alice can only be paid if she gets the LRL-update from LI’s website, and then installs this LRL-update on her device. If this installation is successful, Alice’s DRM client will generate and send LI Msg1_{RDC} which assures LI that Alice’s resold license has been revoked. As LI trusts Alice’s DRM client, and if the verifications, LI_{V7} through to LI_{V9}, are all positive, LI will pay Alice an amount of money equal to the payment, Payment_B. Therefore, once Alice resells (makes a deal and revokes her license), she will get paid, thus addressing the requirement (A2).

A successful execution of the RDC protocol enables LI to be assured that Alice can no longer use the license, Lic. This is because an LRC message, i.e. Msg1_{RDC}, can only be created by Alice’s DRM client which is a tamper-proof software. Hence, it is hard for Alice to forge the LRC message. Therefore, when LI has received the LRC message from Alice, LI can be assured that the license, Lic, has been revoked on Alice’s device, thus satisfying (L4).

5.5 Threat and Attack Analysis

In this section, we discuss the possible threats and attacks that may be mounted against our proposed solution (FSLRP). These threats and attacks are identified based on the requirement set in Section 5.2.3 and the in-depth analysis of the possible threats and attacks in the license reselling context Section 5.5.1 and Section 5.5.2 discuss the threats and the attacks, respectively. They also introduce countermeasures for these threats and attacks.

5.5.1 Threats Analysis

There are two potential threats to our proposed solution (FSLRP). This section analyses these threats and discusses measures that are taken to counter these threats.
5.5. THREAT AND ATTACK ANALYSIS

5.5.1.1 Double Use of a License

Double use of a license refers to a scenario where two consumers, e.g. a reseller and a buyer, are able to use the same license simultaneously. In our case, by a successful execution of the RDA protocol, Bob will receive an activated license, $\text{Lic}_{\text{Activated}}$, before this license is revoked on Alice’s device, i.e. before executing the RDC protocol. In this case, Alice and Bob will have a period of time (between a successful execution of the RDA protocol till a successful execution of the RDC protocol) during which both of them can use the same license (i.e. double use of one license). This would violate the content owner’s rights.

One could say that during the process of issuing this resalable license to the reseller, LI has received a payment, $P_1$, from the reseller. Also, during the execution of the RDA protocol, LI has received another payment, $P_2$, from the buyer of this license. In other words, LI has got two payments for one license, one from the original buyer (i.e. the reseller), and the other from the second buyer. Thus, LI should not complain if the reseller and the buyer use the same license for a period of time. This is true, but LI has to forward $P_2$ to the reseller at the end of a successful execution of the RDC protocol. Therefore, LI has to make sure that this double use is not going to happen.

A solution to address this problem is for LI to issue the buyer, Bob, a date-based license. A date-based license is one that is activated on a specific date. This license can be issued to Bob after a successful execution of the RDA protocol with LI. Upon receipt of this license, Bob’s DRM client will not allow Bob to use this license till the time/date matches with the activation time/date specified. The activation time/date is set to the day following expiry of the deadline, $RD_{DL_A}$, at which Alice’s license should have been revoked. Before the expiry of $RD_{DL_A}$, LI should execute the RDC protocol with the reseller, Alice, to revoke Alice’s resold license. As discussed in Section 5.1.3.3 during the execution of 2M-RDS protocol, Alice and Bob should have agreed to complete the reselling process before an RD deadline which is divided into two deadlines, $RD_{DL_B}$ and $RD_{DL_A}$.

Of course, it does not make sense for Bob to pay for a license and receive it but then be able to directly access it on his device. However, this may be the price to pay as after all, Bob will get a cheaper license in comparison with the same license which is offered by LI.
5.5.1.2 Installing an Out-of-date LRL-update

An out-of-date LRL-update, \( LRL_{\text{Old}} \), is an LRL which does not include the ID of the license being resold (i.e. the one that should be revoked). As discussed in Section 5.4.3 to get paid by LI, a reseller should download, install the latest LRL-update on his device and then send LI a message confirming a successful installation.

A reseller, Alice, may attempt to cheat LI by installing \( LRL_{\text{Old}} \) on her device. Upon a successful installation of this \( LRL_{\text{Old}} \), Alice’s license will not be revoked, because \( LRL_{\text{Old}} \) will not revoke Alice’s resold license. On the other hand, since the installation is successfully done, Alice’s DRM client will generate \( \text{Meg}1_{\text{RDC}} \) and send it to LI. Consequently, Alice can claim for the payment from LI while she can still be able to use the resold license, \( Lic \). Nonetheless, by performing Verification \( LIV_8 \), LI can detect that Alice has installed an out-of-date LRL-update. Also, LI will not send any payment to Alice. Thus, by installing \( LRL_{\text{Old}} \), Alice could not gain anything useful. Furthermore, LI can add Alice to a blacklist to prevent her from further reselling licenses.

5.5.2 Attack Analysis

In this section, we discuss two attacks which may be mounted on our proposed solution (FSLRP).

5.5.2.1 Collusion Attack 1

A collusion attack is one in which Bob and Alice collude to cheat LI. To form this attack both Alice and Bob have to collude. Bob may accept an invalid \( RP_{\text{Lic}} \) (e.g. \( \text{EnDate}_{RP} \) is expired) of Alice’s license while Alice offers Bob a very cheap price for her license. Then both could agree upon a very cheap deal to trade the license. Then, by executing our proposed solution (FRLRS), (i.e. Bob makes the payment to LI and sends an RD activation request), Bob will obtain a non-resalable license. Alice, on the other hand, will receive this payment, agreed in the RD, from LI should she confirm to LI that her license, \( Lic \), has been revoked. In this case, the content owner’s rights would be violated as a license with an invalid \( RP \) has been resold, i.e. non-resalable license has been resold.

To counter this attack, the RP is constructed with a license reselling validity period, i.e. it contains a \( \text{Start Date} \) \( (\text{StDate}_{RP}) \) and an \( \text{End date} \) \( (\text{EnDate}_{RP}) \).
5.5. THREAT AND ATTACK ANALYSIS

Prior to activating RD sent by Bob, as explained in Section 5.4.2, LI checks whether \( R_{P_Lic} \) is still valid (i.e. by performing \( L_{I_{V4.2}} \)). If the validity period of \( R_{P_Lic} \) has expired, LI terminates the reselling process, thus preventing both Alice and Bob colluding to cheat LI.

### 5.5.2.2 Collusion Attack 2

Another collusion attack could be mounted as follows. Alice and Bob may agree on a very cheap deal for a license, so as to pay a small amount to LI. As described in Section 5.1.3.3, the RD should have a validity deadline before which Bob should make an agreed payment to LI to collect the license and Alice should confirm with LI that she has revoked her license to collect the payment made by Bob. If the RD has a single validity period for both Alice and Bob, the following cheating may be possible. Alice may sign a very cheap deal ONLY with Bob. In other words, Alice does not offer her license to any other buyer except Bob. Upon getting a signed RD with Alice, Bob may delay sending the RD activation request to LI until the last minutes of the RD validity deadline. Bob then, by following the FSLRP suite, will get a fresh non-resalable license from LI. It is now difficult for LI to obtain an LRC message from Alice. If LI requests Alice to revoke the license, she, by colluding with Bob, may refuse to revoke his license as the validity deadline of the RD is expired. In addition, Alice may receive an amount of money from Bob as compensation for offering Bob a very cheap deal for \( Lic \). Therefore, Alice can still continue to use her license by simply refusing to revoke it. Now both Bob and Alice can use the same license. In this case, LI will be unable to protect the content owner’s rights.

The above attack has been countered by dividing the RD validity deadline into two deadlines: \( RD_{DLA} \) and \( RD_{DLB} \). Before \( RD_{DLB} \) expires, if Bob wants to proceed with the deal, he should make the payment and send LI an RD activation request. Also, before \( RD_{DLA} \) expires, to collect the payment from LI, Alice should install an LRL-update to allow her DRM client to send LI an LRC message. If Alice refuses to install the LRL-update, she will not get payment from LI. In addition, as Alice has put her signature (see Section 5.4.1) on her deadline \( RD_{DLA} \), she cannot falsely deny having agreed to revoke her license by \( RD_{DLB} \). This allows LI to impose a charge on Alice if she has refused to revoke her resold license, \( Lic \).
CHAPTER 5. RESELLING DEAL METHOD

5.6 FSLRP suite Evaluation

In this section, we evaluate the FSLRP suite. This evaluation is conducted in terms of computational costs and a comparison with related work.

5.6.1 FSLRP Suite Computational Cost

In this section, the computational cost of the FSLRP suite is evaluated. As the exponentiation operations, Exp#, are the heaviest computational operations in the designed protocol suite, the evaluation will be largely performed by computing the number of the exponentiation operations used in the protocol. Table 5.1 shows the number of exponentiation operations performed by each entity during an execution of an FSLRP suite.

Table 5.1: Exponentiation operations performed by FSLRP participants

<table>
<thead>
<tr>
<th></th>
<th>Alice Exp#</th>
<th>Bob Exp#</th>
<th>LI Exp#</th>
</tr>
</thead>
<tbody>
<tr>
<td>During 2M-RDS</td>
<td>6</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>During RDA</td>
<td>0</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>During RDC</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Total Exp#</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

From Table 5.1, it can be seen that an execution of the FSLRP suite requires 30 exponentiation operations, Exp#. These operations are performed by the participants (Alice, Bob, and LI) as follows. Alice performs 9 Exp# (during an execution of the 2M-RDS and RDC protocols). She performs 6 Exp# when executing a 2M-RDS protocol with Bob (1 Exp# in the setup algorithm, 2 when ambiguously signing an RD deal, 3 when verifying Bob’s ambiguous signature). In addition, Alice performs 3 Exp# during a running of an RDC protocol with LI (1 when verifying LI’s signature on $LRL_{Update}$, 1 when signing a License Revocation Confirmation (LRC) message, and 1 when verifying LI’s signature on the payment message).

In an execution of an FSLRP suite, Bob performs 10 Exp# (during an execution of 2M-RDS and RDA protocols). During an execution of a 2M-RDS protocol, Bob performs 8 Exp# (1 in the setup algorithm, 1 when verifying LI’s signature on the permission).

Note that, prior to a license reselling process, i.e. during issuing a resalable license for the first time, LI performs one Exp# when signing a (single or multiple) reselling permission; and Alice also performs one Exp# when verifying LI’s signature on the permission.
5.6. **FSLRP SUITE EVALUATION**

on RP, 1 when verifying LI’s signature on Lic−File, 2 when ambiguously signing the RD deal, 3 when verifying Alice’s ambiguous signature). When running an RDA protocol, he does 2 Exp# (1 when signing RD−Act−Req, and 1 when verifying the result of RD−Act−Req received from LI).

In an execution of an FSLRP suite, LI performs 11 Exp# (when executing the RDA and RDC protocols). During an execution of an RDA protocol, LI does 9 Exp# (1 when verifying Bob’s signature on RD−Act−Req, 1 when verifying RP included in RD−Act−Req, 6 when verifying Alice’s and Bob’s ambiguous signatures, and 1 when sending Bob an activated license which require LI’s signatures). LI also performs 2 Exp# when executing an RDC protocol with Alice (1 when creating a signature on LRLUpdate, and 1 when verifying Alice’s signature on an LRC message).

From the discussion above, it can be noticed that LI will be costed 11 exponentiation operations to facilitate secure and at the same time fair license reselling. As a reward for doing these operations, LI could establish a new business model distinguishing between a resalable and non-resalable license.

### 5.6.2 Comparison with Related Work

In this section, we compare our proposed solution (FSLRP) with the related work presented in chapter 3 (i.e. Kwok’s system [61], Sun’s system [62], NPGCT system [63], Nuovo system [64], Conrado’s system [50], and Laila’s system [65]). The features used in the comparison are summarised as follows:

- **Protecting content owner’s rights, i.e.**
  - Preventing continued use: A reseller cannot continue using a resold license.
  - Preventing reselling a non-resalable License: A reseller cannot resell a non-resalable license.
  - Preventing unauthorised reselling: A reseller cannot resell a resalable license an unauthorised number of times.
  - Support buyer’s traceability: A license issuer can trace the identity of a buyer, who has bought a second-hand license, if this buyer has violated the licensee’s usage rules.
- **Support non-repudiation:** Once a license reselling is done, a license issuer can obtain irrefutable evidence that a reseller has indeed resold his license.

- **Support fairness:** A license reselling process is fairly achieved (i.e. at the end of a reselling process, either the reseller receives a payment and the buyer receives a license, or neither receives anything useful.

- **Abuse-freeness:** A license reselling process is conducted such that during the execution of the process, neither reseller nor buyer could get anything that can be used to prove to an outside entity that any one of them is in control of the outcome of the RD signing process.

- **Support reseller’s and buyer’s interests:** This includes:
  
  - **Monetary interest:** A buyer and a reseller are allowed to maximise their respective monetary interests (i.e. a buyer could pay as little as possible for a second-hand license; and a reseller could gain as much as possible for reselling his license.
  
  - **Do not add additional cost:** An additional cost is not added to a reseller and a buyer to resell and buy a second-hand license, respectively.
  
  - **License authenticity:** A buyer can verify that a license he is about to purchase is indeed issued by an authentic LI.
  
  - **Resale-ability check:** A buyer can verify that a license he is about to buy is resalable.

- **Other features:** These features include:
  
  - **Play content online:** Should a consumer play a content online to be able to resell a license of this content?
  
  - **Use license issuer:** In a reselling process, a license issuer’s role could be either online, or, offline, or not involved at all.
  
  - **Use trusted hardware:** A reselling process requires the use of trusted hardware.
### Table 5.2: Comparison between our solution (FSLRP) and related work

<table>
<thead>
<tr>
<th>Feature</th>
<th>Kwok</th>
<th>Sun</th>
<th>NPGCT</th>
<th>Nuovo</th>
<th>Conrado</th>
<th>Laila</th>
<th>Our solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Differentiate between resalable and non-resalable licenses</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Play content online</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Resell original license</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Prevent continued use</td>
<td>Yes*</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Prevent unauthorised reselling</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>License authenticity</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Buyer traceability</td>
<td>No</td>
<td>Yes</td>
<td>Yes**</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Use trusted hardware</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes/No*</td>
<td>No</td>
</tr>
<tr>
<td>Add cost to consumers</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes/No*</td>
<td>No</td>
</tr>
<tr>
<td>Support fair reselling</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Support non-repudiation</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Re-saleability check</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Support consumers’ monetary interest</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Support abuse-freeness</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>License issuer’s role</td>
<td>No</td>
<td>online</td>
<td>No</td>
<td>offline</td>
<td>online</td>
<td>online</td>
<td>online</td>
</tr>
</tbody>
</table>

* Yes, if Laila’s system is implemented with trusted hardware, and No if not.

** Yes, if LI is involved in the payment process, No if not.
The results of the comparison are shown in Table 5.2. From this table, it can be seen that our solution is the only reselling solution that supports a differentiation between resalable and non-resalable license, license re-saleability check, abuse-freeness, and consumer’s monetary interest. The features of the differentiation between resalable and non-resalable license and the re-saleability check are achieved by use of the novel digital token, a Reselling Permission (RP). A resalable license comes with RP and a non-resalable one without RP. When a reselling process is conducting, through a permission, RP, a buyer can ascertain that a license he is about to buy is resalable before he engages in a purchase process. The features of abuse-freeness and consumer’s monetary interest are accomplished by integrating the CS scheme with the existing license distribution infrastructure (i.e. LI). With these features, our solution has a more profound implication in a marketplace; it offers a three-win facility. For LI, the reseller actually acts as a proxy seller helping LI to reach out to a broader market. The reseller is able to use and then resell his license at the highest possible price and the buyer can buy a cheap second-hand license at the lowest possible price. This is achieved by the fair and abuse-free RDS protocol.

The table also shows that only our solution and the Nuovo solution support license reselling with fairness and non-repudiation properties. However, the two solutions differ in approach. The Nuovo solution makes use of a trusted hardware-based approach, while our two follow a software-based approach. Using the trusted hardware helped the Nuovo system to achieve properties such as preventing an unauthorized reselling, and keeping LI’s overheads at a minimum level. Nonetheless, as mentioned earlier, using trusted hardware normally imposes an additional cost on both reseller and buyer. In addition, the Nuovo system has not considered the scenario and implications of a reseller wanting to resell the original license once he has resold N copies of it. These implications include (1) how to prevent a reseller from reselling the original license an unauthorised number of times, (2) how to prevent a reseller from continuing to use a resold license. In our solution, trusted hardware is not required. Instead, the solution makes use of a contract signing protocol (i.e. RDS protocol described in chapter 4), a digital token (RP), and an assistance role from LI (i.e. the existing license distribution infrastructure). The RDS protocol is used to bind a reseller and a buyer to a reselling process. The RP is used to indicate that a license is resalable.
The assistance role played by LI in our solution is a simple extension of the role already played by LI in the current DRM systems. For example, LI can, during the process of issuing an original license, embed RP into this license to make it resalable. In addition, payment processing, license activation and revocation are all part of the functions already performed in the process of distribution original licenses. Therefore, we can claim that, with the use of cryptographic primitives, our solution supports a license reselling facility, and this novel facility is built on the existing license distribution infrastructure. In this manner, our solution does not add additional cost to consumers while providing fairness and non-repudiation.

Moreover, the results of the comparison, shown in the table, demonstrate that our solution and Laila’s solution are the only proposal that do not use trusted hardware. However, as discussed in chapter 3, Lalia’s solution suffers from two problems not scalable (i.e. it does not support tracks and video contents) and the consumers have to contact a trusted party to play/view their digital contents (i.e. it is not convenient for consumers).

Like Sun’s, Liali’s, and Conrado’s solutions, our solution requires a License Issuer (LI) to be online during an execution of a reselling process. LI helps in preventing continued use and an unauthorised reselling of a license after it has been resold. These two features can easily be achieved in Sun’s and Liali’s solutions as users of these solutions are supposed to use an online service to play/view their content. In Conrado’s system, a special hardware-based license revocation mechanism is used whereas in our solution, we only make use of software-based revocation mechanisms. Thus, our solution is more cost-effective. On the other hand, in Kwok’s and NPGCT’s solution, LI is not involved at all as these solutions make use of a full-trusted hardware based approach.

As with all other solutions, except Kwok’s, our solution enables LI to collect a buyer’s identity during a license reselling process. Thus, if the buyer has violated usage rights of a second-hand license, LI can trace this buyer’s identity.

5.7 Chapter Summary

This chapter has presented a novel solution (FSLRP) to support secure license reselling with a fairness property. Supporting this reselling facility by FSLRP also

5If it is implemented with the use of software mechanisms.
allows both resellers and buyers to maximise their relative monetary benefits. To achieve these properties, the FSLRP design has made use of (1) the CS scheme, and (2) the existing license distribution infrastructure (i.e. LI). The CS scheme provides two interesting properties: (a) weak fairness in supporting exchange of digital signatures and ambiguity in identifying the signers of signatures. By combining the weak fairness property of the CS scheme with the License Issuer (LI) in the existing license distribution infrastructure, our FSLRP suite achieves strong fairness with non-repudiation property. Also, by utilising the ambiguity property of the CS scheme, FSLRP allows resellers and buyers to search for a deal that best suite their respective monetary interests, thus achieving the aforementioned benefits. In addition, by embedding a Reselling Permission (RP) token within a license, the buyer can check whether this license is resalable before he engages in a reselling process with the reseller of this license. Moreover, the use of a software-based mechanism (i.e. LRL) to revoke (terminate) the use of a resold license by its reseller makes the FSLRP cost-effective. The FSLRP analysis has shown that our solution satisfies the design requirements, and can thwart potential threats and attacks in a process of digital license reselling. Furthermore, the evaluation of the FSLRP suite has been conducted and it has demonstrated that our solution is cost-effective and supports new features (supporting monetary interest and differentiating between resalable and non-resalable licenses).
Chapter 6

Two Methods Supporting Multi-Reselling of Digital Licenses

6.1 Chapter Introduction

This chapter presents two methods to support Multi-reselling of a Digital License (MRoDL). The first method is called Repeated Reselling Permission based Multiple Reselling (RRP-MR). This RRP-MR method is a straightforward extension of the RD method (described in chapter 5). This method allows a buyer to control whether a multiple reselling can be continued or stopped. It also allows LI (i.e. content owner) to gain more profit with every reselling of a license. The second method is known as Hash Chain based Multiple Reselling (HC-MR). The HC-MR method minimises the amount of overhead placed on LI as the result of supporting multiple license reselling. It also allows a buyer to check how many times a license can be further resold.

This chapter is organised as follows. Section 6.2 gives an overview of the building blocks used in the design of the two methods. Section 6.3 presents the design preliminaries of the two methods, namely definitions, notations, assumptions and requirements. Section 6.4 describes the RRP-MR method and its analysis. The HC-MR method and its analysis are presented in Section 6.5. Section 6.6 compares the two methods with each other and with related work. Finally, Section 6.7 summarises the chapter.
6.2 Additional Design Building Block

The design of the two license multiple reselling methods makes use of a number of building blocks. They are a digital signature (described in Appendix B), 2M-RDS protocol (described in chapter 5), License revocation list (described in chapter 5), existing license distribution infrastructure (described in chapter 5), and hash chains. The remaining part of this section describes the additional building block (the hash chains).

Hash chains as a cryptographic primitive were first introduced by Lamport [138]. Lamport et al used the hash chains to implement the one-time password (OTP) authentication scheme. As hash chains are low-cost techniques, in addition to authentication [138, 139], Lamport finds many other applications including micropayments [140, 141, 142], certificate revocation [143, 144], online auctions [145], digital cash [146], and one time signature schemes [147].

A hash chain of length N is generated by applying a one-way hash function to an initial random number, \( X_0 \), iteratively. This process outputs a chain of the form: \( X_1 = H(X_0); \ X_2 = H(X_1); \ldots; \ X_N = H(X_{N-1}) \). The last hash value \( X_N \) is also called the tip, \( T \), of the hash chain. \( X_N, X_{N-1} \) cannot be generated by anyone unless he knows the value \( X_0 \). However, given \( X_{N-1} \), its correctness can be verified using \( X_N \). This property of hash chains is inherited from the property of one-way hash functions [138].

In most of the hash chain applications, the last hash value, \( X_N \), is first securely distributed or used. Then starting from the value \( X_{N-1} \), the other values of the hash chain are used one by one till the value, \( X_0 \), is reached.

6.3 Preliminaries

This section gives the license type and digital token definitions, and design assumptions and requirements of the RRP-MR and HC-MR methods.

6.3.1 Definitions

In addition to single resalable licenses, we have identified three types of Multiple Resalable (MR) licenses.

Definition 1: MR Type I (MR-I) License
6.3. PRELIMINARIES

MR-I license can be resold N times by N different consumers. Suppose that a consumer, $C_1$, owning an N-time resalable license, can resell it to another consumer, $C_2$. Then, $C_2$ can also resell it to a third consumer, $C_3$, and so on, until the number of times this license has been resold is N. Once a reseller (e.g. $C_1$ or $C_2$) has resold this MR-I license once, he cannot reuse nor resell it again. In other words, for this type of licenses, a reseller can only resell it once. Every time the license is moved from one consumer to another, a reselling counter associated to this license is decremented by one until it reaches zero at which point the license can no longer be resold.

**Definition 2: MR Type (MR-II) License**

A multi-resalable license may be a MR-II license which can be resold N times by only one consumer, $C_1$. This means that (a) after each reselling of the license and provided that the number of times the license has been resold is less than N times, $C_1$ will still be able to use it on his device and able to resell it; (b) once this license is resold N times, $C_1$ can no longer resell or reuse it. There is one restriction with this MR-II type license. That is, any consumer, who has bought a copy of this MR-II license from another consumer, will not be able to resell it again.

**Definition 3: MR Type (MR-III) License**

An MR-III license can be resold N times by M consumers where $N \geq M$. In other words, a consumer, $C_1$, owning an N-time MR-III license, can resell it to another consumer, $C_2$ with the right for $C_2$ to resell this license again $X$ times, where $X < N$. Both of $C_1$ and $C_2$ can use and/or resell this license as long as the upper limit, N and X, respectively, do not reach zero. Once both of X and N become zero, the owner of the license can use it but can no longer resell it.

**Definition 4: Single Resalable (SR) License**

An SR license can only be resold once. Once a buyer has bought this license from a reseller, it becomes a non-resalable one, so the buyer cannot resell it again.

The scope of this thesis is to support secure and fair resellings of SR and MR-I types of licenses. Owing to the time limitation, we leave the task of supporting MR-II and MR-II license reselling to future work.

**Definition 5: Multiple Reselling Permission**

A Multiple Reselling Permission (MRP) is a digital token which consists of a license identity, an active keystone fix, N (an upper limit of the number of times the license can be resold), and LI’s signature on these items. It is issued with a
given license to authorise this license to be resold N times. In other words, \( MRP \) makes an associated license a multi-resalable license for N times.

**Definition 6: Active Keystone Fix**

An Active Keystone Fix (AKF) is a keystone fix of an \( MRP \) permission. It is needed in the latest reselling process of a license specified in the MRP. This keystone fix is a hash value of a random number called keystone. This keystone is created by LI and shared between LI and the authorised license reseller.

### 6.3.2 Design Assumptions

The following assumptions are used in the design of the two methods, RRP-MR and HC-MR:

1. LI is a trustworthy entity that issues fresh multi-resalable (MR), single-resalable (SR) and non-resalable (Non-RS) licenses to consumers. It also plays an assistant role during a reselling process. It verifies the license legitimacy, activates a license on a buyer’s device, and deactivates the license from a reseller’s device. It also receives a payment from a buyer and sends it to a reseller.

2. A reseller has got either an SR license or MR license with \( RP_{Lic} \) or \( MRP_{Lic} \), respectively.

3. Hash functions are secure, one-way and collision free.

4. Communication channels among entities are resilient,

5. Communication channels are authenticated, confidential, and integrity-protected. These security properties can be obtained by using the Secure Socket Layer (SSL) protocol.

### 6.3.3 Design Requirements

The RRP-MR and HC-MR methods are designed to satisfy the following requirements:

**R1- Detection of unauthorised multi-reselling:** Any unauthorised reselling of a (single or multiple) resalable license should be detected.
R2- **License integrity protection:** Given an N-time resalable license, it should be possible for a consumer and LI to detect whether the upper limit, N, of this license has been tampered with.

R3- **Minimal overhead on LI:** While providing the MRoDL facility, any additional overhead on LI should be kept at a minimum level.

R4- **Multi-reselling check:** The recipient (i.e. a buyer, or LI) of an N-time resalable license should be able to verify how many times the license can further be resold.

R5- **License authenticity:** The recipient of a license should be able to confirm the authenticity of the license (i.e. whether this license and its MRP or RP are indeed issued by LI).

### 6.4 Method One: Repeated RP based Multi-reselling

The RRP-MR (Repeated Reselling Permission based Multi-reselling) method is designed to support multiple resellings of a License. This method is extended from the RD method that has been described in chapter 5. It differs from the RD method in that with the RD method, a buyer obtaining a resold license cannot resell, i.e. the license is non-resalable. On the other hand, with the RRP-MR method the buyer obtaining a resold license can resell it but only once, i.e. the license is an SR license.

The main idea of the RRP-MR method is to allow a buyer of a second-hand license to resell it again. As described in the RD method in chapter 5, a buyer, once obtaining a signed deal, needs to send LI an RD activation request to activate the signed deal. This is necessary to activate the purchased license on his device. With the RD method, the activated license will be a non resalable one. However, by adding some extension to the RD method, we can allow the buyer to request for a resalable license when a second-hand license is being activated. This can be achieved by having LI issue a new RP with the activated license provided that the buyer has paid an extra fee for this RP. Upon the receipt of this RP with the activated license, the buyer can later use RP to resell the license again. The multi-reselling of this license can continue in this way as long as a buyer asks LI
for a new RP with the license being activated. In other words, this multi-reselling process will stop when a particular buyer does not wishes to obtain a resalable license.

6.4.1 RRP-MR Method Overview

The overall process of reselling a SR license, Lic, N times using the RRP-MR method is illustrated in Figure 6.1. The method starts when a consumer, Reseller1, wants to resell his SR − Lic to another consumer, Buyer1. Reseller1 and Buyer1 first negotiate and sign a deal, RD1. Buyer1 then (1) requests LI to activate RD1, and (2) requests LI to make Lic single-resalable again. Thus, when Buyer1 gets Lic, he can further resell it to another buyer, Buyer2. To resell Lic for the 2nd time, Buyer1 will act as Reseller2. In the 2nd reselling process, Reseller2 and Buyer2 first negotiate and sign another deal, RD2. Then, similar to the case in the 1st reselling process, Buyer2 requests LI to activate RD2 and to make Lic resalable as well. This process repeats until the Nth reselling process is completed, at which point the license obtained from LI is a non-resalable one.

6.4.2 RRP-MR Method in Detail

As illustrated in Figure 6.3, the RRP-MR method consists of four phases: RD creation, RD signing, RD activation-request, and RD activation.

**RD Creation Phase:** A reseller and a buyer negotiate and agree on a deal, known as RD, for the license reselling/purchasing process. This RD contains the license identity, the price for this license, and the terms and conditions of the license reselling process.

**RD Signing Phase:** To sign the negotiated RD, the buyer first performs two verifications (License Authentication, LA), an (Re-Saleability (RS) Check) (the LA and RS checks are detailed in Section 6.4.3). If these two checks are positive, the buyer will execute the 2M-RDS protocol (described in chapter 5), with the reseller to fairly sign the RD negotiated in the RD creation phase. The buyer then initiates the RD activation-request phase with LI.

**RD Activation-request Phase:** In this phase, the buyer requests for RD activation and also requests LI to make the license, Lic, as resalable. This is done
Figure 6.1: N resellings of a license, Lic, using the RRP-MR method
CHAPTER 6. TWO METHODS SUPPORTING MULTI-RESELLING

Figure 6.2: The outline of the RD method

by having Bob launch the RDA protocol\footnote{The RDA (Reselling Deal Activation) protocol is described in chapter 5.} with modification in the RD activation-request. As illustrated in Figure 6.4, the RD activation-request message comprises the following items: Signed RD, License RP, Readability Flag, Payment, and Buyer’s Signature.

- **Signed RD**: This is a reselling deal which has been signed in the RD signing phase. It contains the license identity, agreed price, and terms and conditions.

- **License RP**: This is the reselling permission for the license being resold.

- **Re-saleability Flag (RF)**: This is a flag indicating the type of license the buyer has requested for. The flag has two values: (1) ReSalable (RS), or (2) Non-ReSalable (Non-RS). When it is set to RS (i.e. $RF = 1$), LI will send Bob a SR license. If it is set to Non-RS (i.e. $RF = 0$), LI will send Bob a Non-RS license.
Payment: This is the amount of money the buyer ought to pay to have his purchased license activated by LI. There may be different payment amounts. For example, if the amount is equal to the agreed price stated in the deal, RD, this may indicate that the license to be activated is a Non-RS license. However, for an SR license, the payment may be higher than the agreed price specified in RD as LI may impose a service of re-saleability fee. The fee is the amount of money paid to LI to make Lic resalable again.

Buyer’s Signature: This is the buyer’s signature which is signed on all the above items to protect the integrity of the RD activation request.

Once Bob has generated the RD activation request, he sends it to LI to invoke the RD activation-request phase (i.e. step 3 in Figure 6.3). Upon the receipt of this request, as depicted in Figure 6.9, LI will perform a number of verifications known as LI’s verifications, \( LI_{V_1\text{-to-}V_6} \).

RD Activation Phase: In this phase, one of two procedures is performed. Firstly, if the RF flag in the RD activation request is set to Non-RS, LI will perform the RD activation procedure for Non-RS licenses. This procedure has
been described in chapter 5. Secondly, if the flag is set to RS, as depicted in Figure 6.3, LI will perform the RD activation procedure for RS licenses, which is described below:

- **Issuing a new RP to the activated License, Lic**: This is achieved by:
  
  1. Generating a new keystone, $ks_2$, and this will be shared with Bob.
  
  2. Generating and signing a new reselling permission for Lic (i.e. $RP_{2,Lic}$). $RP_{2,Lic}$ is of the form, $RP_{2,Lic} = [Lic||f_2||Sig_{LI}(Lic||f_2)]$, where $f_2$ is the keystone fix of $ks_2$, i.e. $f_2 = H(ks_2)$.

- **Activating Lic on Bob’s device**: This is done by:
  
  1. Assigning a new ID to Lic to avoid the license ID problem mentioned in chapter 5;
  2. Encrypting Lic and $RP_{2,Lic}$ using Bob’s public key;
  3. Sending the encrypted message to Bob

- **Marking the license with $RP_{1,Lic}$ as resold**: Once the license attached with $RP_{1,Lic}$ is resold, LI needs to revoke $RP_{1,Lic}$, so as to protect the license from any unauthorised reselling in the future. One way of doing this is for LI to host a Read-only Public Directory (RPD). As illustrated in Figure 6.5, LI publicises the keystone, $ks_1$, contained in $RP_{1,Lic}$, on this RPD. Publishing $ks_1$ on RPD allows LI (through verification, $LI_{V4,3}$) to detect if $RP_{1,Lic}$ is used again to resell the license. In addition, any buyer, using the Active Keystone Fix (AKF) check, can check this RPD to confirm that (a)

\[\text{Figure 6.4: The structure of an RD activation request of the RRP-MR method}\]
a reselling permission, e.g. $RP_{2_{Lic}}$, is still fresh (i.e. it has not been used in a previous reselling of the same license) before proceeding in the reselling process.

- **Revoking Lic on Alice’s device**: LI revokes the resold license, $Lic$, on Alice’s device. This can be done by using the RDC protocol described in chapter 5.

- **Sending payment to Alice**: For the activation of a resalable license reselling, LI will deduct the re-saleability fee from the payment received from Bob and send the rest of the amount to Alice. This amount should be equal to the price specified in the deal, $RD$, being activated. For activating a non-resalable license, LI will send Alice the payment received from Bob. This can also be accomplished during the execution of the RDC protocol described in chapter 5.

![Figure 6.5: LI’s RPD after reselling Lic 2 times](image)

Once $RD_1$ is activated, Bob should have got the license and $RP_{2_{Lic}}$. This means that Bob can use the license on his device and resell it again to a new buyer using $RP_{2_{Lic}}$. If, for example, Charlie wants to buy this license from Bob, Charlie and Bob will use $f_2$ and $k_{s2}$ to sign a new deal ( $RD_2$ ). In the RD activation-request phase of $RD_2$, to obtain a resalable copy of the license, Charlie has to set the RF flag to RS again. While activating $RD_2$, LI will issue a new reselling permission, $RP_{3_{Lic}}$, to Charlie. Hence, Charlie can use $RP_{3_{Lic}}$ to resell the license for the 3rd time. The multi-reselling process of the license can continue in this way till the $N^{th}$ buyer sends LI an RD activation request with RF=0 (i.e.
the flag RF is set to Non-RS). In this case, the $N^{th}$ buyer will get a non-resalable license which can not be resold by the buyer.

It can be seen that, with the RRP-MR method, any of the buyers will have the power to stop or to continue the multi-reselling process of a license. If the buyer wants to get a resalable license, he has to pay LI an extra fee and set the RF flag to RS in the RD activation request (see step 3 in Figure 6.3). In this case, LI will issue a new $RP$ for the license and send it to the buyer. Thus, the buyer can resell the license again (i.e. continue the multi-reselling process). If the buyer does not want to resell the license in the future, he can just set the resalablity flag to Non-RS (non-resalable). LI will then send the buyer a non-resalable license and the multi-reselling process of this license will stop.

6.4.3 Verifications used in the RRP-MR Method

This section describes all the verifications used during the execution of the RRP-MR method. These verifications include checks and verifications performed by both a buyer and LI. Prior to signing an reselling deal, RD, with a reseller, the buyer performs the following verifications, i.e. Re-Saleability (RS) check, License Authenticity (LA) verification, and Active Keystone Fix (AKF) check. In addition, before activating a signed RD, LI performs the verifications: $LI_{V1}$ through to $LI_{V6}$.

6.4.3.1 Buyer’s Verifications

Re-saleability (RS) Check: A buyer uses this check to verify whether a license he is about to purchase is resalable.

The detail of the RS check is illustrated in Figure 6.6. It consists of two further verifications. In the first verification, Bob verifies that equation (6.1) holds, where $Lic_{RD_{AB}}$ is the license ID contained in $RD_{AB}$ negotiated by Alice and Bob, and $Lic_{RP1_{Lic}}$ is the license contained in $RP1_{Lic}$.

$$Lic_{RD_{AB}} = Lic_{RP1_{Lic}}$$  \hspace{1cm} (6.1)

This first verification ensures that the reselling permission granted to the license indeed belongs to the license. In the second verification, as shown in Figure 6.7, Bob verifies that LI’s signature, $Sig_{LI}(Lic\|f)$, on $RP1_{Lic}$ is indeed valid. A valid signature proves that $RP1_{Lic}$ has not been tampered with after its
6.4. **METHOD ONE: REPEATED RP BASED MULTI-RESELLING**

Bob will stop the reselling process. If both of them are positive, it means that \( \text{Lic} \) is resalable. Thus, Bob can proceed to sign RD for this license with Alice.

**License Authentication (LA) Verification:** This verification is designed for a buyer to verify that a given license, \( \text{Lic} \), and its associated \( \text{RP}_{\text{Lic}} \) are indeed issued by LI.

RS check verifies that a reselling permission is indeed issued by LI. To confirm that a license is certainly issued by LI, as illustrated in Figure 6.8, LI’s signature on the license, \( \text{Lic} \) i.e. \( \text{Sig}_{\text{LI}}(\text{Lic}) \) is verified. If the RS check and the verification of \( \text{Sig}_{\text{LI}}(\text{Lic}) \) are positive, the buyer will be assured that the license, \( \text{Lic} \), and its reselling permission, \( \text{RP}_{1\text{Lic}} \), are indeed issued by LI (i.e. they are both authentic).

**Active Keystone Fix (AKF) Check:** This check allows the buyer to verify that a keystone fix, provided in a reselling permission, is fresh (i.e. has not been used yet). The AKF check is performed by the buyer before he signs a deal, RD, with a reseller. If the check is negative, the buyer will not invoke a signing process. For example, suppose that a buyer wants to check that a keystone fix, \( f \), contained in a permission, \( \text{RP}_{\text{Lic}} = [\text{Lic}||f||\text{Sig}_{\text{Lic}}(\text{Lic}||f)] \), is fresh. To perform the AKF check, the buyer performs the following operations:

(a) Get the keystone fix, \( f \), from the permission, \( \text{RP}_{\text{Lic}} \). Let us denote this keystone fix as \( f_{RP} \);

![Figure 6.6: The RS check](image-url)
(b) Retrieve the corresponding keystone fix from the AKF field of LI’s RPD and denote it as $f_{RPD}$:
(c) Verify that the equation (6.2) holds.

\[ f_{RPD} = f_{RP} \]  

(6.2)

If the verification in (c) is negative, it means that the keystone fix contained in \( RP_{lic} \) is not valid for this reselling process. The buyer should then terminate the reselling process. If this verification is positive, it means that the keystone is valid, and the reselling process should proceed.

6.4.3.2 LI’s verifications

LI performs a number of verifications before starting the RD activation process. These verifications are designed to protect content owners’ rights. They enable LI to detect (1) the reselling of non-resalable license; (2) unauthorised
reselling of a license. A license reselling is authorised if and only if the following verifications are all positive.

\((LI_{V1})\): Confirm that the buyer has made the due payment and verify the correctness of Bob’s signature on the RD activation request.

\((LI_{V2})\): Check the value of the re-saleability flag. If it is RS, LI performs \(LI_{V3}\). Otherwise, LI performs \(LI_{V4}\).

\((LI_{V3})\): Verify the amount of the received payment. This verifications comprises two further verifications, \(LI_{V3,1}\), and \(LI_{V3,2}\):

- In \(LI_{V3,1}\), LI confirms that the payment made is equal to the amount stated in the signed RD plus the RS fee.
- In \(LI_{V3,1}\), LI checks that the payment made is equal to the amount stated in the signed RD only.

\((LI_{V4})\): Check that the license, \(Lic\), is valid and resalable. This check consists of three further checks: \(LI_{V4,1}\), \(LI_{V4,2}\), and \(LI_{V4,3}\):

- In \(LI_{V4,1}\), LI confirms that \(Lic\) has a valid \(RP_{Lic}\). This is done by performing LI’s signature on \(RP_{Lic}\). If this verification is not positive, it means that \(RP_{Lic}\) has been modified. \(Lic\) is then deemed as non-resalable. LI will then reject RD. Otherwise, \(Lic\) is considered as resalable and LI proceeds to perform \(LI_{V4,2}\).
- In \(LI_{V4,2}\), LI ascertains that the license ID in \(RP_{Lic}\) and in the signed RD are identical. This check prevents a reseller or a buyer from replacing \(RP_{Lic}\) with another less valuable reselling permission. If \(LI_{V4,2}\) is positive, LI will perform \(LI_{V4,3}\). Otherwise, LI stops the reselling process.
- In \(LI_{V4,3}\), LI verifies that \(RP_{Lic}\) has not been used yet. This is done by checking whether the value of the keystone, corresponding to keystone fix given in the \(RP_{Lic}\) exists in the Used Keystone field of the RPD. If yes, it means that this \(RP_{Lic}\) has already been used. LI then stops this reselling process. If this keystone does not exist in the LI’s RPD, then \(RP_{Lic}\) has not been used yet. LI then proceeds to perform \(LI_{V5}\).

\((LI_{V5})\): Verify the buyer’s signature on RD to confirm that the buyer has signed RD.
(LI\textsubscript{V6}): Verify the reseller’s signature on RD to ensure that the reseller has signed RD.

If the re-saleability flag is RS and if all these verifications are positive, LI will proceeds to the RD activation phase to activate Lic as a resalable license.

6.4.4 RRP-MR Method Analysis

This section gives an analysis of the RRP-MR method against its requirements set out in Section 6.3.3. It also discusses potential attacks on the method and describes countermeasures for these attacks.

6.4.4.1 Analysis against Requirements

Detection of unauthorised reselling: The RRP-MR method is designed such that any unauthorised reselling can be detected. An unauthorised reselling may be attempted in any of the following ways: (a) using a reselling permission granted to a SR license twice, (b) reselling a non-resalable license, or (c) reselling a resalable license with an invalid reselling permission. A reseller may perform an unauthorised reselling to cheat a buyer. Also, a buyer may attempt an unauthorised reselling to cheat LI. However, with the AKF check, the buyer can detect attempt (a). Additionally, with the LA and RS verifications, the buyer can detect attempts (b) and (c), respectively. Using the verifications, LI\textsubscript{V4.1}, LI\textsubscript{V4.2}, and LI\textsubscript{V4.3}, LI can detect any unauthorised reselling committed by the buyer using (a), (b), and (c), respectively. More details about unauthorised reselling attacks are given in Section 6.4.4.2.

License integrity protection: With the RRP-MR method, both a resalable license and its RP are integrity-protected using digital signatures. During the issuance stage of the license and its RP, LI digitally signs both of them. Therefore, if any of them has been modified, during the verifications of LA ( or LI\textsubscript{V4.1}), a buyer (or LI) can detect this modification, respectively.

Re-saleability check: Using the RRP-MR method, a buyer is enabled to check whether a license he is about to purchase is resalable using the RS verification and whether it has already been resold using the AKF check. Both checks have been described in Section 6.4.3. Prior to signing a deal, a buyer first performs these two checks. He only proceeds to sign the deal, if both of these two checks are positive.
6.4.4.2 Analysis against Potential Attacks

There are two potential attacks that could be mounted on the RRP-MR method: Unauthorised Reselling (UR) attack, and replay attack. This section describes these attacks and discusses how the RRP-MR method counters these attacks.

Unauthorised Reselling (UR) Attacks We say that a reselling process is under an UR attack if one of the following cases takes place:

1. No-RP: A license reselling is being attempted without a valid reselling permission. That is, the reselling is attempted without any permission or with an incorrect permission.

2. Double-used RP: A license is being resold using a permission that has already been used in a previous reselling of this license. For example, suppose that a license, Lic, has already been resold using $RP_{Lic}^{1}$. When a reselling of Lic is being attempted again using $RP_{Lic}^{1}$, we call this an unauthorised (or double) reselling. This is because $RP_{Lic}^{1}$ has already been used in a previous reselling of Lic.

3. Unmatched-RP: A resalable license, Lic1, that should be resold using $RP_{1}$, is being attempted to be resold using $RP_{2}$. This $RP_{2}$ should be used in reselling the license, Lic2. A motivation for this attack is that a reseller or a buyer may want to replace a cheaper license with a more expensive one, or they may want to use the $RP$ of a resalable license to resell a non-resalable one.

The RRP-MR method has taken a number of measures to counter these UR attacks.

UR Attack by a reseller: The verifications RS, LC, and AKF are designed to detect and thwart any UR attacks mounted by a reseller. In the No-RP attack case, the buyer, using the LA verification, can detect that a given license is not resalable (i.e. has no valid RP). As discussed in Section 6.4.3 the LA verification takes $RP$ as an input. If there is no $RP$, the buyer will terminate the reselling process.

In the Double-used $RP$ attack case, the buyer, using the AKF check, can detect that a given $RP$ has already been used in a previous reselling. As explained in Section 6.4.3 if the equation $6.2$ does not hold, then this $RP$ has already been
used in a previous reselling. This means that the reseller is trying to double use the RP. The buyer can then stop the reselling process.

In the Unmatched-RP attack case, using the RS check, a buyer can detect that a license and the associated RP do not match. For example, suppose that Alice has got a license, Lic1, with RP\textsubscript{Lic1}. This Lic1 can be used for another 10 years and it can be resold for £10 as a second-hand license. At the same time, Alice has also got another license, Lic2, with RP\textsubscript{Lic2}. Lic2 can be further used for 20 years and it can be resold for £15 as a second-hand license. In other words, Lic2 is more valuable than Lic1. Furthermore, suppose that Alice and Bob have negotiated a deal, RD\textsubscript{2}, for the license, Lic2. In RD\textsubscript{2}, they agreed on £15 as the price for the reselling of Lic2. When signing RD\textsubscript{2}, instead of sending RP\textsubscript{Lic2} as evidence that Lic2 is resalable, Alice may send Bob RP\textsubscript{Lic1} which is valid for the license, Lic1 but not for the license, Lic2. By doing this cheating, Alice takes advantage of RP\textsubscript{Lic2} of the higher-priced license, Lic2, to resell a lower-priced Lic1. If this happens and if Bob signs RD\textsubscript{2}, Bob will be cheated as he may pay a higher price for the license, Lic1, which is not worth this price. Also, this is considered as an unauthorised reselling as Lic1 can only be resold using RP\textsubscript{Lic1}.

**UR Attack by a buyer:** A buyer may also mount an UR attack to cheat LI by following any of the above cases. The verification, LI\textsubscript{V4.1}, allows LI to detect the No-RP attack case, the Verification, LI\textsubscript{V4.3}, enables LI to detect the Double-used RP attack case, and the verification, LI\textsubscript{V4.2}, enables LI to detect the Unmatched-RP attack by verifying if a given license identity provided in both RD and RP are identical.

From the above analysis, we can conclude that the RRP-MR method enables the detection of various cheating acts or attacks launched by a reseller or by a buyer.

**Replay Attack**

By using the SSL protocol to establish confidential and integrity-protected communication channels amongst the involved entities (see assumption 5), replay attacks on the protocol messages of the RRP-MR method can be countered. Using the SSL protocol, the protocol messages are not exposed to any outsiders. This prevents any intruder from gaining anything useful from the messages while
they pass through the network.

6.4.4.3 RRP-MR Method Weaknesses

Although supporting multi-reselling by using the RRP-MR method is simple and straightforward, the method suffers from two weaknesses: a fair amount of additional overheads are imposed on LI, and there is a lack of a multi-reselling check.

**Additional overheads on LI:** With the RRP-MR method, LI needs to issue a new reselling permission, $RP$, for every license, $Lic$, being resold. For each $RP$ issuance, LI has to perform the following operations:

1. Generate a random number, i.e. a keystone $ks$,
2. Generate a hash value from $ks$ to obtain a keystone fix, $f$,
3. Construct a new $RP$, $RP_{Lic}$;
4. Generate a signature on $RP_{Lic}$.

Therefore, to resell a license, $Lic$, $N$ times, LI will need to perform this set of four operations $N$ times. This results in a fair amount of overheads to be added on LI, thus making the RRP-MR method inefficient.

**Lack of Multi-reselling Check:** With the RRP-MR method, a buyer cannot check how many times a license can be further resold. The buyer can only verify whether the license is authorised to be resold for the current reselling. This is because the reselling permission used in this method can only authorise the license to be resold once. This means that the RRP-MR method does not support the multi-reselling check.

6.5 Method Two: Hash Chain based Multi-reselling

The HC-MR method is proposed to address the two weaknesses of the RRP-MR method described above. That is, it is designed to provide a multi-reselling facility while minimising the level of overheads placed on LI, and enabling a multi-reselling check by a buyer. Two ideas are used in the design of the HC-MR method: (1) a novel hash-chain based $MRP$ (Multiple Resalable Permission) token is proposed to reduce the level of involvement by LI in a reselling process;
6.5. **METHOD TWO: HASH CHAIN BASED MULTI-RESELLING**

and (2) an online cross checking facility is used to allow buyers to confirm a current MRP token is valid for a current reselling process. With the use of MRP, LI does not have to generate and sign a new RP each time a license is being resold. It will only need to perform two tasks. Firstly, when initially issuing a MR licence, LI generates an MRP token and attaches it to the license. This token contains the permissions for the resellings of the license and it is signed only once by LI at the time when it is first generated. It will then be used in the 1st reselling of the license. This MRP token will later be used by LI to generate further (N-1) MRPs to be used for (N-1) resellings but without LI having to generate any more signatures on these MRPs. This MRP token, as shown in Figure 6.10, is generated by making use of a hash chain primitive. A hash chain of length N is first generated from a root keystone, ks. The last two hash values (i.e. \( h_{N-1} \) and \( h_N \)) are then used as a keystone and a keystone fix, respectively, for the 1st reselling of the license. In this reselling, \( h_N \), as the keystone fix for the 1st reselling, will be included in the 1st MRP and \( h_{N-1} \) will be shared with the 1st reseller as a keystone fix for the 1st reselling. In the 2nd reselling, \( h_{N-1} \), as the keystone fix for the 2nd reselling, will be added to the 2nd MRP and \( h_{N-2} \) will be shared with the 2nd reseller. The process continues until \( ks \) is used in the Nth reselling.

The HC-MR method also supports a multi-reselling check by a buyer. Including the N reselling limit in the 1st MRP which is signed by LI, a buyer can find out how many more resellings can be performed on this license. In the remaining (N-1) resellings, the buyer can use the information published on LI’s RPD along with the data contained in the \( x^{th} \) MRP (where \( x \) is the number of times the license has been resold so far -including the current reselling) to calculate how many times the license can be further resold.
CHAPTER 6. TWO METHODS SUPPORTING MULTI-RESELLING

\[ h_2 = H(h_1) \]
\[ h_1 = H(ks) \]
\[ h_3 = H(h_2) \]
\[ h_4 = H(h_3) \]
\[ h_5 = H(h_4) \]

**Chain Generation**

**Chain Use**

- **h4** and **h5** are shared between LI and 1st Reseller, and used as **ks** and **f** in 1st re-sale of Lic
- **h3** will be only released after 2nd re-sale of Lic
- **h2** will be only released after 3rd re-sale of Lic
- **h1** will be only released after 4th re-sale of Lic
- **ks** is kept secret and only released after 5th (final) re-sale of Lic

**Figure 6.10: Using hash chain to design MRP**
6.5. METHOD TWO: HASH CHAIN BASED MULTI-RESELLING

6.5.1 HC-MR Method Overview

A reselling process using the HC-MR method is illustrated in Figure 6.11. The method consists of four phases: RD Creation, RD Signing, RD Activation-request, and RD Activation. The first three phases are the same as those discussed in the RRP-MR method. By the end of a successful execution of phase 3, (1) LI should have received a payment from the buyer; and (2) both the buyer and the reseller are officially committed to a signed deal, RD. In phase 4, LI starts the RD Activation process in which, as shown in Figure 6.11 LI performs the following tasks. LI first releases the keystone, $h_{N-i}$, where $N$ is the reselling upper limit for the license, Lic, and $i$ is the order of the current reselling. LI then generates a new MRP (i.e. $MRP(i+1)_i$) to be used for the $(i+1)^{th}$ reselling. LI also activates Lic on the buyer’s device. Finally, LI revokes Lic from the reseller’s device, and sends the reseller the payment received from the buyer in the RD activation-request phase. From this point on, Lic can be further resold (N-i) times.

![Figure 6.11: The outline of the HC-MR method](image-url)
6.5.2 Multiple Reselling Permission (MRP)

This section gives a detailed description of MRP which is the main component in the HC-MR method. It explains when and how MRP is generated. It also shows how MRP is linked to a license, Lic, to make it multi-resalable.

As illustrated in Figure 6.12, the association between a given license, Lic, and its MRP is achieved by using a data structure containing:

- Lic: A license identity,
- f: A keystone fix,
- N: The maximum number of times Lic is allowed to be resold,
- SigLI: LI’s signature on the concatenation of the above items to protect the integrity of the association.

An MRP for a license, Lic, is generated at one of two stages: (1) when Lic is first being issued, and (2) when Lic is being resold. When Lic is being issued for the first time, LI generates the first MRP for Lic (i.e. MRP1Lic). MRP1Lic is used for the 1\textsuperscript{st} reselling of Lic. At the i\textsuperscript{th} reselling of Lic, where i = 1, \ldots, N-1, LI will generate MRP(i+1)Lic to be used for the (i+1)\textsuperscript{th} reselling. In fact, the generation of MRP(i+1)Lic, as illustrated in Figure 6.12, is done by only adding a new keystone fix (i.e. hash value) into MRP(i)Lic. The generation process of MRP at these two stages is described below.

Generating MRP when Lic is first being issued (i.e. Generating MRP1Lic): MRP1Lic is the 1\textsuperscript{st} MRP for a license, Lic. It authorises Lic to be resold N times. To generate MRP1Lic, LI performs the following operations:

- Apply a one-way cryptographic hash function to a root keystone, ks, iteratively to generate a chain of hash values. The length of this chain is equal to N, the maximum number of resellings allowed for Lic. The root keystone, ks, will only be released at the last reselling of Lic. For example, when N=5, the hash values, $h_1$ is computed as $h_1 = H(ks)$, and other hash values are computed such that $h_i = H(h_{i-1})$, for $2 \leq i \leq N$.

- LI then uses $h_N$ (the last hash value of the chain) to generate the reselling permission for the 1\textsuperscript{st} reselling which is of the form: $MRP1_{Lic} = [Lic||h_N||N||SigLI(Lic)||h_N||N]$ , where:
6.5. METHOD TWO: HASH CHAIN BASED MULTI-RESELLING

\( Lic \): is the identity of the multi-resalable license;

\( N \): is the maximum number of the times \( Lic \) can be resold;

\( h_N \): is the \( N^{th} \) (i.e. the last) hash value of the root keystone, \( ks \). \( h_N \) is the hash value of \( h_{N-1} \) which should be securely sent to a consumer (the reseller in our case). The hash values \( h_N \) and \( h_{N-1} \) will be used as a keystone fix and a keystone, respectively, in the 1\(^{st} \) license reselling.

\( \text{Sig}_{LI}(\text{Lic}||h_N||N) \): is LI’s signature on \( MRP_{1_{\text{Lic}}} \). This signature is to prove that \( MRP_{1_{\text{Lic}}} \) is indeed generated by LI and the associated \( \text{Lic} \) can be resold \( N \) times.

Once \( MRP_{1_{\text{Lic}}} \) is generated, LI will send it along with \( \text{Lic} \) to the 1\(^{st} \) reseller completing the process of issuing the multi-resalable license, \( \text{Lic} \). In addition, LI will securely send the hash value, \( h_{N-1} \), where \( h_N = \text{H}(h_{N-1}) \), to the 1\(^{st} \) reseller for it to be used as a keystone in the 1\(^{st} \) reselling. This can be accomplished by encrypting \( h_{N-1} \) with the reseller’s public key. The other hash values \( (h_{N-2}, \ldots, h_1, ks) \) will be kept secret by LI to be used in future resellings.

Note that, by adding the number of resellings, \( N \), to \( MRP_{1_{\text{Lic}}} \), any verifier of \( MRP_{1_{\text{Lic}}} \) can confirm that \( MRP_{1_{\text{Lic}}} \) is authorising \( \text{Lic} \) to be resold \( N \) times. In addition, as \( N \) is included in LI’s signature, no one can modify (e.g. increase) the value of \( N \) without being detected. Thus, by constructing the permission token, \( MRP_{1_{\text{Lic}}} \), where \( MRP_{1_{\text{Lic}}} = [\text{Lic}||h_N||N||\text{Sig}_{LI}(\text{Lic}||h_N||N)] \), any buyer can verify whether the permission is authentic. The buyer can also perform the multi-reselling check before he engages in a reselling process of the license.

**Generating \( MRP(i+1)_{\text{Lic}} \) at the \( i^{th} \) Reselling of \( \text{Lic} \):** At the \( i^{th} \) reselling of \( \text{Lic} \), where \( i=1, \ldots, N-1 \), LI should also give the \( i^{th} \) buyer a multiple reselling permission, \( MRP(i+1)_{\text{Lic}} \), authorising \( \text{Lic} \) to be resold (\( N-i \)) times after this current reselling. To upgrade \( MRP(i)_{\text{Lic}} \) into \( MRP(i+1)_{\text{Lic}} \), the only step LI needs to do is to add the hash value, \( h_{N-i} \), into \( MRP(i)_{\text{Lic}} \) to form \( MRP(i+1)_{\text{Lic}} \), where \( h_{N-i} \) was the keystone for the \( i^{th} \) reselling. \( h_{N-i} \) will be used as the keystone fix in the \((i+1)^{th}\) reselling.

Assuming \( N = 5 \), the process of upgrading \( MRP(i) \) is as follows. At the 1\(^{st} \) reselling of \( \text{Lic} \), LI will do the following operation to generate \( MRP_{2_{\text{Lic}}} \) from \( MRP_{1_{\text{Lic}}} \).

- As illustrated in Figure 6.12 b, LI adds the keystone of the 1\(^{st} \) reselling, i.e. \( h_4 \), to the AKF field of \( MRP_{1_{\text{Lic}}} \). The output of this addition is \( MRP_{2_{\text{Lic}}} \).
which is of the form $MRP_{2Lic} = [Lic||h_4||h_5||5||Sig_{LI}(Lic||h_5||5)]$. Although $h_4$ is not signed by LI, anyone can still verify that it is generated by LI. The idea is that, as (a) $h_5 = H(h_4)$; (b) $h_5$ is signed by LI in $Sig_{LI}(Lic||h_5||5)$; (c) $h_4$ is contained in $MRP_{2Lic}$; (d) $h_4$ is made public, as discussed in Section 6.5.4, $h_4$ must have been generated by LI. Therefore, by using the hash chain primitive to get a hash chain of the root keystone, $ks$, and by only signing the last hash value, $h_N$, in $MRP_{1Lic}$, the integrity of all the hash values in the chain can be protected. Thus, LI only needs to sign a single signature for generating N MRPs for $Lic$.

Figure 6.12 shows the structure of 5 MRPs of $Lic$ generated (a) during issuing $Lic$, and (b) during the 4 times of reselling $Lic$. It can be seen from this figure...
that LI only signs $MRP_{Lic}$, thus reducing the overhead imposed on LI.

### 6.5.3 Read-only Public Directory (RPD) for Multi-Reselling Checking

If a particular permission, $MRP(i)_{Lic}$, for a MR license, $Lic$, has been used there should be a way allowing any future buyers to be informed of this fact, i.e. $MRP(i)_{Lic}$ has already been used and $MRP(i+1)_{Lic}$ is the next valid permission to be used. This can be done by having LI publicise $h_i$ which is the keystone of the $i^{th}$ reselling of $Lic$. One way to make $h_i$ public is to publish it on a Read-only Public Directory (RPD). RPD, as illustrated on Figure 6.13, consists of two fields: license identity ($Lic$), and hash values associated to $Lic$. These hash values are grouped into two sub-fields: Used Keystone Fix (UKF), and Active Keystone Fix (AKF). The UKF field contains the keystone fixes already used in previous reselling processes (i.e. the keystone fixes were contained in used MRPs). The AKF field contains the keystone fix that is valid to be used next (i.e. it is contained in the current MRP). Figure 6.13 illustrates the structure of RPD of a license, $Lic$, at different resellings.

Using RPD, any buyer can check whether a given keystone fix has been used in a previous reselling. Also, as discussed in Section 6.5.5, using the AKF check, a buyer can verify whether a keystone fix is the active/right one to be used in the current reselling.

### 6.5.4 HC-MR Method in Detail

The HC-MR method, as illustrated in Figure 6.11, consists of four phases: RD creation, RD signing, RD activation-request, and RD activation. These phases are described below.

**RD Creation Phase:** In the RD creation phase, a reseller and a buyer negotiate and agree on a mutually acceptable deal, known as RD. This RD contains terms and conditions of a reselling process of a license. In addition, it includes the license identity and an agreed price for this license.

**RD Signing Phase:** To sign the negotiated Reselling Deal, RD, the buyer first performs two verifications License Authentication (LA), and Multi-reselling Check (MrC). As discussed in Section 6.5.5, if these two checks are positive, the buyer will execute with the reseller the 2M-RDS protocol (described in chapter
to sign the RD negotiated. The buyer then initiates the RD activation-request phase with LI.

**RD Activation-request Phase:** This phase is designed for the buyer to ask LI to activate RD which is signed in the RD signing phase. The buyer also makes the payment stated in RD to LI. In other words, the buyer invokes the RDA protocol (described in chapter 5) and sends an RD activation request consisting of the signed RD, agreed payment, and $MRP(i)_{Lic}$, where $i$ is the order of the reselling. Upon the receipt of this request, LI will perform the verifications, depicted in Figure 6.21, and described in Section 6.5.5. If all these verifications are positive, LI will proceed to the RD activation phase.

**RD Activation Phase:** In this phase, LI will perform two tasks: (a) sending the buyer an activated multi-resalable license, and (b) sending the reseller the
payment received from the buyer. To send the multi-resalable license, LI will perform the following operations:

- **Release** \( h_{N-i} \): This is to mark \( Lic \) as resold using \( MRP(i)_{Lic} \).

- **Create** \( MRP(i + 1)_{Lic} \): This is to make \( Lic \) as \((N - i)\) times resalable license. As discussed in Section 6.5.2, generating \( MRP(i + 1)_{Lic} \) does not require LI’s signature, but \( MRP(i + 1)_{Lic} \) will still be integrity-protected. With the facility of RPD, the property of hash chain, and LI’s signature on the root of the hash chain, \( MRP(i + 1)_{Lic} \) can still be used to prove the authenticity of the \((i + 1)^{th}\) reselling. From this, future buyers can find out how many times \( Lic \) can further be resold.

- **Activate** \( Lic \) on the buyer’s device: This is done by:
  1. Having LI assign a new ID to \( Lic \) to avoid the license ID problem mentioned in chapter 5;
  2. Having LI encrypt \( Lic \) using the public key of the buyer’s device, and
  3. Having LI send \( Lic \) and \( MRP(i + 1)_{Lic} \) to the buyer.

To send the reseller the payment, LI will do the following operations:

**Revoke** \( Lic \) on the reseller’s device: This can be achieved by using the RDC protocol presented in chapter 5.

**Send the reseller the payment:** Once LI has received the confirmation that \( Lic \) has been revoked on the reseller’s device, LI sends the payment received from the buyer in the RD activation-request phase to the reseller.

Once the RD activation phase is successfully executed, the buyer will receive an \((N-i)\) resalable \( Lic \), so he can resell it again if he wants to. The reseller will also receive the payment for his resold license, \( Lic \).

**An exemplar illustration of using the HC-MR Method**

The following scenario illustrates how the HC-MR method is used to resell a license, \( Lic \), 2 times out of \( N=5 \) times. The scenario will show (1) how \( MRP \) can be used 5 times without being re-signed by LI at every reselling process; and (2) how a buyer can use this \( MRP \) to check how many times \( Lic \) can further be resold at or after each reselling. In this scenario, there are three consumers involved. As
illustrated in Figure 6.14, Alice will resell Lic to Bob in the 1\textsuperscript{st} reselling process. Then, in the 2\textsuperscript{nd} reselling process, Bob will resell Lic to Charlie.

The 1\textsuperscript{st} reselling of Lic: In this reselling, Alice, the 1\textsuperscript{st} reseller, will sell her multi-resalable license, Lic, to Bob, the 1\textsuperscript{st} buyer. It is assumed that Alice has got a 5-time resalable license, Lic, from LI prior to the 1\textsuperscript{st} reselling process. This means that Alice has got Lic, h4, and $MRP_{1 Lic} = [Lic | H_5 | 5 || \text{Sig}_{Lic}(Lic | H_5 | 5)]$. In the 1\textsuperscript{st} reselling of Lic, as illustrated in Figure 6.14, Alice and Bob perform the following operations:
6.5. METHOD TWO: HASH CHAIN BASED MULTI-RESELLING

1. They negotiate a deal called $RD_{AB}$. $RD_{AB}$ contains the terms and conditions of the 1st reselling and the price for $Lic$.

2. They sign $RD_{AB}$ using the RDS protocol described in chapter 4. In this signing process, Alice will use $h_4$ as the keystone, and $h_5 = H(h_4)$ as the keystone fix.

3. Once the deal $RD_{AB}$ is signed, Bob will submit it along with the agreed payment to LI for $RD_{AB}$ activation. Prior to activating the license on Bob’s device, LI will perform the verifications, $LI_{V1}$ through to $LI_{V5}$, which are discussed in Section 6.5.5 and depicted in Figure 6.21.

4. If all these verifications are positive, in the RD activation phase, LI will perform the following operations:

   (a) **Publicising $h_4$ on RPD:** To mark the license, $Lic$, as resold once, LI will release $h_4$ which is the keystone used in the 1st reselling of $Lic$. After its publication, $h_4$ can not be used as a keystone for any future resellings. However, as discussed in Section 6.5.2, as $h_4$ is equal to $H(h_3)$, $h_4$ will be used as a keystone fix for $h_3$ in the 2nd reselling.

   (b) **Creating the 2nd reselling permission for $Lic$ (i.e. $MRP_{2 Lic}$):** This is done by upgrading $MRP_{1 Lic}$ to $MRP_{2 Lic}$. As discussed in Section 6.5.2, this upgrading does not involve any new signature generations, but rather only requires the addition of a new item, i.e. $h_4$, into $MRP_{1 Lic}$, resulting $MRP_{2 Lic}$. The $MRP_{2 Lic}$ will allow $Lic$ to be resold 4 times.

   (c) **Activating $Lic$ on Bob’s device:** This is accomplished by:
      - Having LI assign a new ID to $Lic$ to avoid the license ID problem mentioned in chapter 5;
      - Having LI encrypt $Lic$ using Bob’s device public key;
      - Having LI send the encrypted license, $Lic$, and $MRP_{2 Lic}$ to Bob.

   (d) **$Lic$ Revocation:** LI must perform the revocation of $Lic$ on Alice’s device to prevent Alice from using $Lic$ after she has resold it. This can be achieved by using RDC protocol described in chapter 5.

---

3 This is achieved during the execution of the RDA protocol described in chapter 5.
(e) Sending the Payment to Alice: LI finishes off the 1st reselling by sending Alice the payment received from Bob in the RD activation-request phase. Like the case of Lic revocation, this can also be accomplished during the execution of the RDC protocol described in chapter 5.

Upon the successful execution of all these operations, Bob should have now received the license, Lic, and its reselling permission, MRP2Lic. This means that Bob has received a multi-resalable license which can be further resold 4 times. In addition, Alice should have received the payment agreed in RD and will not be able to access Lic on her device anymore.

The 2nd Reselling of the license, Lic: In this reselling, Bob, as the reseller, will resell Lic to the 2nd buyer of the license, Charlie. As shown in Figure 6.14, similar to the case in the 1st reselling of Lic, Bob and Charlie will first negotiate a deal, known as RDBC. They then sign RDBC using the RDS protocol described in chapter 4. In the signing process, $h_3$ and $h_4 = H(h_3)$ will be used as a keystone and keystone fix, respectively. Charlie then invokes the RD activation-request phase with LI. In this phase, she sends LI the payment stated in RDBC along with an activation request for RDBC. After receiving this request, LI will perform the verifications, $LI_{V1}$ through to $LI_{V5}$, illustrated in Figure 6.21. If the verifications are all positive, LI will perform the following operations to activate the license on Charlie’s device:

1. Publicising $h_3$ on RPD: To mark Lic as resold twice, LI will release $h_3$ which was used as the keystone for the 2nd reselling. This is done in the same way as publicising $h_4$ on RPD. Now, as illustrated in Figure 6.5c, LI’s RPD contains three hash values, $h_5$ and $h_4$ which are in the UKF field while $h_3$ is in the AKF field. This means that $h_3$ can be used as a keystone fix in the 3rd reselling.

2. Creating $MRP3_{Lic}$: This is done by upgrading $MRP2_{Lic}$ to $MRP3_{Lic}$. As discussed in the 1st reselling above, this modification does not require LI’s signature on $MRP3_{Lic}$, but rather only includes the addition of a new item, i.e. $h_3$, into $MRP3_{Lic}$, where $MRP3_{Lic} = [Lic||H_3||H_4||H_5||5||Sig_{Lic}(Lic||H_5||5)]$. As explained in creating $MRP2_{Lic}$, although $h_3$ in $MRP3_{Lic}$ is not signed by LI, a recipient of $MRP3_{Lic}$ is still able to verify that $MRP3_{Lic}$ is generated by LI.
3. **Activating Lic on Charlie’s device:** This is accomplished by:

   (a) Having LI assign a new ID to Lic to avoid the license ID problem mentioned in chapter 5;

   (b) Having LI encrypt Lic using Charlie’s device public key. Hence, this Lic can only be accessed on Charlie’s device which holds Charlie’s private key;

   (c) Having LI send the encrypted Lic and its MRP3Lic to Charlie.

4. **Lic Revocation:** Lic on Bob’s device should now be revoked. This is achieved by using the RDC protocol introduced in chapter 5.

5. **Sending the Payment to Bob:** LI finishes off the 2nd reselling by sending Bob the payment received from Charlie during the RD activation-request phase.

   After performing all the above operations successfully, Charlie will receive the license, Lic, and its reselling permission, MRP3Lic. The MRP3Lic will allow Lic to be further resold 3 times. In other words, Charlie will still be able to resell Lic to another buyer. Also, Bob will obtain the agreed payment and will not be able to use Lic anymore.

   The 3rd, 4th, and 5th resellings of Lic will be performed in the same way. After the 5th reselling, LI will not send an MRP to the 5th buyer and will update LI’s RPD such that, as shown in Figure 6.5d, the AKF field on LI’s RPD is Nil. At this stage, Lic will no longer be resalable. In other words, when the number of the hash values in the UKF field of LI’s RPD is equal to N, indicated in MRP5Lic, which is equal to 5 (in this example), the owner of Lic can no longer resell it.

6.5.5 **Verifications Used in the HC-MR Method**

This section describes all the verifications used during the execution of the RRP-MR method. These verifications include those performed by a buyer and by LI. Prior to signing a deal, RD, with a reseller, the buyer performs the verifications, **multi-reselling check**, **active keystone fix check**, and **license authentication verification**. Before activating a signed RD, LI performs the verifications, LI\(_{V1}\) through to LI\(_{V6}\).
6.5.5.1 Buyer’s Verifications

**Multi-reselling check:** This is a check by which a buyer of a given license can verify whether the license is a N-time resalable. This check is performed in two different ways based on whether it is performed at the 1st reselling ($MrC_1$), or at the $i^{th}$ reselling ($MrC_i$), where $2 \leq i \leq N$.

![Diagram](image)

Figure 6.15: The $MrC_1$ check

**Multi-reselling Check at the 1st reselling ($MrC_1$):** $MrC_1$ is illustrated in Figure 6.15. It comprises two further verifications. In the first verification, Bob verifies that equation (6.3) holds, where $Lic_{RDAB}$ is the license ID contained in the negotiated deal, $RD_{AB}$, and $Lic_{MRP1_{Lic}}$ is the license ID contained in $MRP1_{Lic}$.

\[
Lic_{RDAB} = Lic_{MRP1_{Lic}}\tag{6.3}
\]

In the second verification, as illustrated in Figure 6.16, Bob verifies LI’s signature, $Sig_{Lic}(Lic || h_5)$, on $MRP1_{Lic}$. If both these verifications are positive, it means that $NoR_{Lic} = 5$. This is because $NoR_{Lic} = 5 = N$ is signed by LI and the signature is contained in $MRP1_{Lic}$. By this verification, Bob can confirm that $Lic$ is multi-resalable and the maximum authorised number of resellings is 5.

**Multi-reselling Check at the $i^{th}$ reselling ($MrC_i$):** $MrC_i$, where $2 \leq i \leq N$, is designed to allow the $i^{th}$ buyer to check how many times a license, $Lic$, can be further resold after the $i^{th}$ reselling. One may ask why $MrC_1$ cannot be used to verify $NoR_{Lic}$ at the $i^{th}$ reselling. This is because starting from the $i^{th}$ reselling ($i > 1$), $Lic$ can only be resold $(N - i + 1)$ times not N times. The value, $(N - i + 1)$, is not signed by LI in the $i^{th}$ $MRP$ of $Lic$, which is of the form
6.5. METHOD TWO: HASH CHAIN BASED MULTI-RESELLING

\[ MRP(i)_{\text{Lic}} = [\text{Lic}||h_N||h_{N-1}||...||h_{N-i+1}||N||\text{Sig}_{\text{Lic}}(\text{Lic}||h_N||N)] , \text{ where } 2 \leq i \leq N, \text{ and } i \text{ is the number of times the license has been resold so far- including the current one, and its value is equal to the number of hash values provided in } MRP(i)_{\text{Lic}}. \]

Thus, performing \( MrC_1 \) on the \( i^{th} \) \( MRP \) can only verify that the maximum number of times \( \text{Lic} \) can be resold is \( N \). It cannot verify that \( \text{Lic} \) can be further resold \( (N-i+1) \) times. For example, when \( i = 2 \), and \( N = 5 \), the \( 2^{nd} \) \( MRP \) of \( \text{Lic} \) will be of the form \( MRP2_{\text{Lic}} = [\text{Lic}||h_4||h_5||5||\text{Sig}_{\text{Lic}}(\text{Lic}||h_5||5)]. \)

Performing \( MrC_1 \) on \( MRP2_{\text{Lic}} \) can only prove that \( \text{Lic} \) is a resalable license, and it can be resold 5 times in total. However, it cannot verify that \( \text{Lic} \) is allowed to be resold 4 more times after the completion of the \( 1^{st} \) reselling. Hence, \( MrC_1 \) cannot be used to prove to a buyer that \( \text{Lic} \) can only be resold 4 times.

In addition, with this form of \( MRP2_{\text{Lic}} \), any entity (e.g. including its owner) can remove \( h_4 \) from \( MRP2_{\text{Lic}} \) without being detected. This is because \( h_4 \) is not signed by \( \text{Lic} \). Without any additional protection mechanism, it may be possible for the owner of \( MRP2_{\text{Lic}} \) to falsely claim that \( \text{Lic} \) is a fresh 5-time resalable license even if it has been resold once. One benefit this cheating may bring to the owner is that he may resell \( \text{Lic} \) for a higher price.

There are two possible solutions to the latter problem. The first is for \( \text{Lic} \) to add the value \( (N-i+1) \) into \( MRP(i)_{\text{Lic}} \) and to sign his signature on \( MRP(i)_{\text{Lic}} \) at the \( i^{th} \) reselling. With this method, \( \text{Lic} \) needs to sign a digital signature for
every reselling undertaken. As signature generation is an expensive operation, this solution, similar to the RRP-MR method, will add a fair amount of additional overhead on LI. The second solution is to let LI maintains an RPD table publishing the used and active keystone fixes. By using the Active Keystone Fix (AKF) check which is simply a table look-up check, any entity can simply use it to calculate the actual remaining number of resellings (i.e. $NoR_{Lic}$) a license, $Lic$ is authorised for. The HC-MR method uses the second solution.

![Figure 6.17: The MrC$_i$ check](image)

The $MrC_i$ check consists of two verifications and one calculation of $NoR_{Lic}$. As shown in Figure 6.17, the first verification is to confirm that the license ID contained in the negotiated RD, i.e. $Lic_{RD}$, is identical to the license ID contained in $MRP(i)_{Lic}$, i.e. if the following equation holds.

$$Lic_{RD} = Lic_{MRP(i)_{Lic}}$$ (6.4)

The second verification is called Active Keystone Fix (AKF) Check and it is shown in Figure 6.18. The AKF check is designed for a buyer to ascertain that the hash value, $h_{N-i+1}$, provided in $MRP(i)_{Lic}$, where $2 \leq i \leq N$, is the one to be used in the $i^{th}$ reselling. This check makes use of LI’s RPD and $MRP(i)_{Lic}$. A buyer first gets $h_{N-i+1}$ from $MRP(i)_{Lic}$ and then compares it with the hash value published on the AKF field of LI’s RPD. If these two values are equal the verification result is positive. Otherwise, the buyer stops the $i^{th}$ reselling.

Revisiting the example described in Section 6.5.4, in the $2^{nd}$ reselling of $Lic$, in the $2^{nd}$ reselling of $Lic$, ...
6.5. **METHOD TWO: HASH CHAIN BASED MULTI-RESELLING**

Charlie needs to perform the AKF check on

\[ MRP_{2 Lic} = [Lic||H_4||H_5||5||\text{Sig}_{Lic}(Lic||H_5||5)] \].

As shown in Figure 6.18, to perform the AKF check, Charlie does the following operations:

(a) Get \( h_5 \) and \( h_4 \) from \( MRP_{2 Lic} \);

(b) Check that the equation (6.5) holds. If not, Charlie stops the 2\(^{nd}\) purchasing/reselling process with Bob. Otherwise, if yes, Charlie proceeds to perform the operations (c) and (d);

\[ h_5 = H(h_4) \] \hspace{1cm} (6.5)

(c) Retrieve the hash value (e.g. \( h_x \)) from the AKF field of LI’s RPD;

(d) Check that the equation (6.6) holds. If not, Charlie terminates the reselling.
If yes, it means that $h_4$ is the AKF of the $2^{nd}$ reselling.

$$h_x = H(h_4) \quad (6.6)$$

If all the above checks are positive, Charlie can confirm that $h_4$ is AKF for the $2^{nd}$ reselling. She can then proceed to calculate how many times $Lic$ can be further resold using equation (6.7).

**Calculating the Number of Reselling of $Lic$ ($NoR_{Lic}$):** This calculation is done using the following equation:

$$NoR_{Lic} = (N - h_{No}) + 1 \quad (6.7)$$

where:

- $N$ is the upper limit of the number of resellings, $Lic$, has been signed for in LI’s signature, $\text{Sig}_{LI}(Lic || h_N || N)$;

- $h_{No}$ is the number of the hash values listed in $MRP_{2,Lic}$ (i.e. $h_5$ and $h_4$) in this example. $h_{No}$ should be equal to the number of the hash values published on LI’s RPD.

Note that the value of $h_{No}$ given in $MRP_{2,Lic}$ means that the associated license has already been resold ($h_{No} - 1$) times. Using equation (6.7), Charlie can be assured that $Lic$ can still be resold 4 times.

**License Authentication (LA) Verification:** An LA verification is designed to verify that a given license, $Lic$, and its $MRP$ are indeed issued by LI (i.e. authentic). It is performed in two different ways based on the order of a reselling process; at the 1$^{st}$ reselling, $LA_1$ is used, and at the $i^{th}$ reselling, $LA_i$ is used, where $2 \leq i \leq N$. These two ways are explained as follows.

**LA Verification at the 1$^{st}$ Reselling ($LA_1$):** $LA_1$ consists of two signature verifications, as shown in Figure 6.19, (a) verifying LI’s signature on $MRP_{1,Lic}$, and (b) verifying LI’s signature on the license file, $Lic - File$.

**LA verification at the $i^{th}$ Reselling ($LA_i$):** Starting from the $i^{th}$ reselling, where $i=2, \ldots, N$, $MRP(i)_{Lic}$ contains hash values which are not directly signed by LI, but hash-chain linked to the token signed by LI. Thus, applying $LA_1$ alone is not sufficient to prove the authenticity of $MRP(i)_{Lic}$.

Verification $LA_i$ consists of a Hash Chain (HC) check and $LA_1$ check. $LA_1$ check has just been described above. In the HC check, the buyer verifies that
the hash values contained in $\text{MRP}(i)_{\text{Lic}}$ are chained, i.e. $h_N = H(h_{N-1}), h_{N-1} = H(h_{N-2}), \ldots, h_{N-i} = H(h_{N-i+1}), \ldots, h_2 = H(h_1)$. For example, in the example described in Section 6.5.4, Charlie, during the 2\textsuperscript{nd} reselling, performs $LA_2$ to make sure that Lic and $\text{MRP}_2\text{Lic}$ are authentic. In detail, as depicted in Figure 6.20, $LA_2$ involves the following operations:

(a) Confirm that equation (6.8) holds, where $h_5$ and $h_4$ are the two hash values contained in $\text{MRP}_2\text{Lic}$, 

$$h_5 = H(h_4) \quad (6.8)$$

(b) Verify LI’s signature, $\text{Sig}_{LI}(\text{Lic}||h_5||5)$, the 1\textsuperscript{st} MRP;

(c) Verify LI’s signature on the license, $\text{Sig}_{LI}(\text{Lic} – \text{File})$.

If (a) and (b) are both correct, it means that $\text{MRP}_2\text{Lic}$ is indeed issued by LI. As $h_5$ equals $H(h_4)$, and $h_5$ is signed by LI, then $\text{MRP}_2\text{Lic}$ must have been generated by LI. Also, if (c) is positive, then Lic is certainly issued by LI. Note that $\text{Sig}_{LI}(\text{Lic}||h_5||5)$ is the same signature created by LI on $\text{MRP}_1\text{Lic}$. This means that LI had not needed to generate a new signature on $\text{MRP}_2\text{Lic}$.

The process of performing the checks, $LA_3, LA_4, \ldots, LA_N$, is identical to what has been described for $LA_2$ above.
6.5.5.2 LI’s verifications

As discussed in Section 6.5.5, LI performs a number of verifications before activating an RD for a buyer. With these verifications, LI can detect any unauthorised reselling to protect content owners’ rights. LI only activates an RD if the following verifications are all positive.

- \((LI_{V1})\): Confirm that the buyer has made the payment,

- \((LI_{V2})\): Verify that this payment is equal to the amount stated in the signed RD and verify the correctness of Bob’s signature on the RD activation request.

- \((LI_{V3})\): Check that Lic is legitimate to resell. This check consists of three further checks: \(LI_{V3.1}\), \(LI_{V3.2}\), and \(LI_{V3.3}\).
  - In \(LI_{V3.1}\), LI verifies whether a license, \(Lic\), has a valid reselling permission, \(MRP(i)_{Lic}\). This is done by performing the LA verification described in Section 6.5.5.1. If this LA verification is not positive, \(Lic\)
is deemed as non-resalable. LI will reject the deal, RD. Otherwise, Lic is considered as resalable and LI proceeds to perform $LI_{V3.2}$.

- In $LI_{V3.2}$, LI ascertains whether Lic in both $MRP(i)_{Lic}$ and in the signed RD are identical. If not, LI will stop the reselling. This check prevents anyone from replacing $MRP(i)_{Lic}$ with another less valuable $MRP(i)_{Lic2}$. If $LI_{V3.2}$ is positive, LI will perform $LI_{V3.3}$.

- In $LI_{V3.3}$, LI verifies that $MRP(i)_{Lic}$ has not been used yet. This is done by checking if LI’s RPD contains $h_{N-i}$ which is the keystone fix contained in $MRP(i)_{Lic}$. If it does, the verification is negative. LI then terminates this reselling. This is because LI only publishes $h_{N-i}$ on LI’s RPD if $MRP(i)_{Lic}$ has already been used. If LI’s RPD does not contain $h_{N-i}$, LI proceeds to perform $LI_{V4}$.

- $(LI_{V4})$: Verify the buyer’s signature on RD to confirm that the buyer has signed RD.

- $(LI_{V5})$: Verify the reseller’s signature on RD to ensure that the reseller has signed RD.

6.5.6 The HC-MR Method Analysis

This section analyses the HC-MR method against its requirements, which are set out in Section 6.3.3. It also discusses potential attacks on the method and proposes countermeasures to these attacks.

6.5.6.1 Analysis against Requirements

Detection of unauthorised reselling: The HC-MR method provides a solution to detect any unauthorised reselling. An unauthorised reselling could be attempted by using anyone of the following means: (a) by double usage of a single MRP, (b) by reselling a non-resalable license, (c) by reselling a resalable license more than $N$ times, or (d) by reselling a resalable license using a mismatched MRP. Any of these means could be attempted either by a reseller to cheat a buyer or by a buyer to cheat LI. If the reseller does any of them, using the verifications $MrC_1$, $MrC_i$, $LA_1$, and $LA_i$, the buyer can detect it. Also, if the buyer does any of these unauthorised resellings, LI can detected it using (a) $LI_{V3}$, and
(b) LI’s RPD. More detail about unauthorised reselling attacks has been given in Section 6.4.4.2.

License integrity protection: With the HC-MR method, both the multi-resalable license and its MRP are integrity-protected. This is achieved by making use of the digital signature. During the generating process of MRP and its license, as described in Section 6.5.2, LI digitally signs both the license and the associated MRP. Thus, any modification in any of them will be detected during signature verifications described in Section 6.5.5.

In $MRP(i)Lic$, where $2 \leq i \leq N$, the $(N - i + 1)$ reselling times of $Lic$ are not signed by LI. However, with the verification $MrC_i$ described in Section 6.5.5, any verifier can confirm that $MRP(i)Lic$ is indeed issued by LI.

Minimal overhead on LI: Using the HC-MR method, the overhead on LI is minimised. This is achieved by taking the following measures. Firstly, $MRP$ is designed such that LI only generates one signature. This signature is per license for the N reselling checks needed for the license and it can be used on any other
MRPs to be generated during the N resellings of the license. Secondly, we made use of the online cross checking (i.e. LI’s RPD) to ensure the security of the multi-reselling process without putting much overhead on LI. In other words, the design of MRP and the use of RPD can help to move some of the processing workload from LI’s side to consumers’ side.

**Multi-reselling check:** The HC-MR method provides a way by which a buyer is able to check how many times a given license can be further resold. This is accomplished by designing two checks: $MrC_1$ and $MrC_i$, where $i = 2 \ldots N$. $MrC_1$, as shown in Figure 6.15, is used at the 1st reselling to prove that a given license can be resold N times. $MrC_i$ is used at the $i^{th}$ reselling to prove that a license can be further resold $(N - i + 1)$ times. By these two checks, a buyer can confirm at any reselling that a license he is about to purchase is resalable and how many times it can be further resold. The buyer can then decide whether the license is worth its offered price.

### 6.5.6.2 Security Analysis against Potential Attacks

We have identified two types of attacks on the HC-MR method: Unauthorised Reselling (UR) attack and replay attack. This section describes how these attacks may be mounted and discuss what countermeasures are used to protect against them.

**Unauthorised Reselling (UR) Attack**

In the context of multi-resalable licenses, a reselling process is considered unauthorised if one of the following cases occur.

1. **No-MRP:** A license without a MRP is being attempted to be resold;
2. **Reselling Exceeds N times:** A reselling of a license, which has already been resold $N$ times, is being attempted $(N + 1)^{th}$ reselling.
3. **Double-used MRP:** A license with a used MRP is being attempted to be resold again using the same MRP. For example, suppose that Lic has been resold with its permission, $MRP_{2_{Lic}}$. Later, this Lic is being attempted to be resold again with $MRP_{2_{Lic}}$. This is called an unauthorised reselling using double-used MRP.

---

*There is not a well-known or standard list of attacks in the license multi-reselling problem context in the literature. Attacks discussed in this section are identified based on our in-depth investigation and analysis of the problem and various scenarios of cheatings by different protocol entities.*
4. **Unmatched-MRP**: A multi-resalable license, $Lic$ with a permission $MRP_1$ is being attempted to be resold using another permission $MRP_2$, where $MRP_1$ authorises $Lic$ to be resold $N$ times, and $MRP_2$ authorises another license, $Lic_2$ to be resold $M$ times, where $M > N$. For example, a reseller or a buyer may want to replace a cheaper $N$-time resalable license with a more expensive one that is $M$-time resalable. Suppose that Alice has got a license, $Lic_1$, with a reselling permission, $MRP_{1_{Lic_1}}$. This $MRP_{1_{Lic_1}}$ allows $Lic_1$ to be resold $N$ times. Also, suppose that she has got another license, $Lic_2$, with $MRP_{1_{Lic_2}}$ which authorises $Lic_2$ to resold $M$-time, where $M > N$. To make $Lic_1$ more expensive (e.g. with a higher price), Alice could use $MRP_{1_{Lic_2}}$ to prove to a buyer, Bob, that $Lic_1$ is $M$-time resalable. Alice then asks Bob for a higher price. If this happens Bob will be cheated as he may pay a higher price than the real cost of $Lic_1$. This scenario is considered as an unauthorised reselling as $Lic_1$ is only authorised to be resold $N$ times, not $M$ times.

Any of the above forms of UR attacks can be mounted either by a reseller to cheat a buyer or by a buyer to cheat LI. However, as discussed below, the HC-MR method can thwart all these forms of UR attacks.

**UR Attack by a Reseller**: A reseller could mount any of the above forms of UR attacks to cheat a buyer. If the reseller mounts a *No-MRP attack*, using the verifications, $MrC_1$ or $MrC_i$, the $1^{st}$ buyer or the $i^{th}$ buyer can detect that a given license is not resalable. As discussed in Section 6.5.5 these two checks both take an $MRP$ permission as its input to verify whether its associated license is multiple resalable. If there is no MRP, the buyer can not perform this check, so will stop the reselling process.

If a reseller mounts a *Resold N-time attack*, using the AKF check, a buyer can detect that a given license has already been resold $N$ times. This is because, if the license has already been resold $N$ times, the AKF check will be negative as the AKF field of $L_1$’s RPD is Nil.

In the case of a *Double-used MRP attack*, by using the AKF check, as explained in Section 6.5.5 the buyer will get a negative result. As a result, the buyer will terminate the reselling process.

In the case of an *Unmatched-MRP attack*, using the checks, $MrC_1$ or $MrC_i$, the buyer can detect that there is a UR attack. For example, as discussed in Section 6.5.5 in the $1^{st}$ reselling, Bob can use $MrC_1$ check to detect that the
license ID provided in the negotiated RD is not identical to that contained in $MP_1 Lic$. In the same way, in the $i$th reselling, using $MrC_i$, any buyer can also detect whether the license identity stated in a negotiated RD is identical to the one provided in a given MRP. So, with the checks $MrC_1$ or $MrC_i$, an UR attack, mounted using an Unmatched-MRP, can be detected.

**UR Attack by a buyer, Bob:** A buyer may also mount any of the above forms of UR attacks to cheat LI. However, with our method, any such UR attacks can be detected by LI. In the case of No-MRP attack, the buyer could send LI an activation request to activate a license, $Lic$, but does not send $MRP$ of the license. However, with the verification $LI_{V3.1}$, LI can detect that this license has no $MRP$ (i.e. it is not resalable). LI then stops the reselling process.

In a Resold N-time attack, verification $LI_{V3.1}$ allows LI to detect this attack. As this verification is performed before the RD activation process. Once LI detects this attack, he can simply stop the reselling process.

In a Double-used $MRP$ attack, with verification $LI_{V3.3}$, LI can detect that $MRP$ contained in an RD activation request has already been used. LI can then terminate the reselling.

In an Unmatched-MRP attack, verification $LI_{V3.2}$ enables LI to detect that $MRP$ provided in an RD activation request does not match the license being resold. LI can then stop the reselling process.

From the above discussions, it can be concluded that the HC-MR method has built-in measures to thwart UR attacks. These measures are $MrC_1$, $MrC_i$, $LA_1$, $LA_i$ and AKF verifications which are built on the primitives such as digital signatures, hash chains and RPD. These verifications allow a buyer to detect any UR attack mounted by a reseller. Also, using verification $LI_{V3}$, LI can detect any UR attack launched by a buyer.

**Replay Attack**

The HC-MR method can also protect against replay attacks that may be mounted on the exchanged messages. This is achieved by making use of confidential and integrity-protected communication channels amongst the involved entities. These channels can be established by the SSL protocol. Thus, the exchanged data is not exposed to any outsiders. This prevents any intruder from gaining anything useful from the messages while they pass through the network. In addition, the SSL protocol can also protect against any replay attack [19]. Therefore, the RRP-MR method is protected against any replay attack.
6.6 Evaluation of the Two Multi-reselling Methods

The evaluation of the RRP-MR and HC-MR methods are performed by (1) evaluating the computational cost of each method as discussed in Section 6.6.1 and (2) comparing the two methods with related work as presented in Section 6.6.2.

6.6.1 Computational Costs of the Two Methods

In this section, the computational cost of the RRP-MR and HC-MR methods is evaluated. As the exponentiation operations, Exp#, are the heaviest computational operations in the design of the two methods, the evaluation will largely be performed by computing the number of the operations used in the execution of both methods. Table 6.1 shows the number of the exponentiation operations, Exp#, performed by each entity during an execution of the RRP-MR method and the HC-MR method.

Table 6.1: Exp# performed when executing the RRP-MR and HC-MR methods

<table>
<thead>
<tr>
<th></th>
<th>RRP-MR Method</th>
<th>HC-MR Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alice Exp#</td>
<td>Bob Exp#</td>
</tr>
<tr>
<td>During 2M-RDS</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>During RDA</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>During RDC</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Sub-total Exp#</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Total Exp#</td>
<td>31</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1 shows that each execution of the RRP-MR method and HC-MR method requires 31 and 30 exponentiation operations, Exp#, respectively. During an execution of the RRP-MR method, (1) Alice performs 9 Exp# (during an execution of the 2M-RDS and RDC protocols); (2) Bob performs 10 Exp# (when executing the 2M-RDS and RDA protocols); (3) LI performs 12 Exp# (during an execution of RDA and RDC protocol). From Table 6.1 and Table 5.1, it can be seen that prior to a license reselling process, i.e. during issuing a resalable license for the first time, LI performs one Exp# when signing a (single or multiple) reselling permission; and Alice also performs one Exp# when verifying LI’s signature on the permission. These two operations should be added to the computational cost of issuing a resalable license.
be seen that (a) Alice and Bob perform the same number of exponentiation operations when executing the RRP-MR method and the RD method; and (b) LI performs one extra Exp\# with each execution of the RRP-MR method than it does with the RD method. This extra operation is performed when LI generates a new reselling permission for an activated license during an execution of the RDA protocol. In an execution of the HC-MR method, the participants Alice, Bob and LI perform the same number of exponentiation operations which they perform during an execution of the RD method (see Table 6.1 and Table 5.1).

![Figure 6.22: LI’s computational cost when one license is resold 5 times with the methods RRP-MR and HC-MR](image)

As shown in Figure 6.22, when the HC-MR method is used, the computational cost imposed on LI to resell one license 5 times is approximately 8.3% less than the cost imposed on LI when the RRP-MR method is used. This improvement is achieved by making use of a multiple reselling permission (MRP) which does not require LI’s signature during each reselling process of the license.

From the discussion above, we can draw the following remarks. Firstly, the HC-MR method is more efficient than the RRP-MR method. Secondly, the RD method, described in chapter 5, and the HC-MR method have the same computational cost. However, the HC-MR method supports two more features (a) allows a license to be resold N-times with different N-consumers, and (b) allows a buyer to verify how many times a license can be further resold before he engages in a purchase/reselling process.
6.6.2 Comparison with Related Work

To the best of the author’s knowledge, the Nuovo DRM system [64] is the only system seen in the literature that supports multi-reselling of digital licenses. This system allows a consumer (a reseller) not only to buy license to access a specific content, but also to resell/distribute N-copies of this license. To achieve this multi-reselling, as described in chapter 3, the Nuovo system relies heavily upon the use of trusted devices, which execute only their embedded certified rules. The Nuovo system also makes use of a License Issuer (LI) if there is a dispute (e.g., caused by hardware failure) to resolve the dispute and achieve fair reselling. The foremost disadvantage of the Nuovo system is that it is not cost-effective. Every reseller/buyer has to use a tamper-proof hardware device. This would introduce additional cost into the underlying reselling process.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Nuovo system</th>
<th>RRP-MR</th>
<th>HC-MR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support multi-reselling facility</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Support fairness</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Support Non-repudiation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Require trusted hardware</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Add additional cost to consumers</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>LI’s overhead</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Support resalable and non-resalable licenses</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi-reselling check</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>License integrity</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Can a reseller resell an original license?</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Can a reseller resell N-copies of a license?</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Can a buyer resell a license again?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Can a buyer detect reseller’s cheating?</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Can a buyer stop a multi-reselling?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Support consumers’ monetary interest</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Computational cost (Exp#)</td>
<td>LI not used</td>
<td>6</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>LI used</td>
<td>12</td>
<td>31</td>
</tr>
</tbody>
</table>

A comparison between our two methods (RRP-MR and HC-MR) and the Nuovo system is performed. This comparison is summarised in Table (6.2). First of all, our two methods and the Nuovo system support a multi-reselling facility with fairness and non-repudiation properties. However, they use two different approaches. The Nuovo system makes use of a trusted hardware-based approach,
6.6. EVALUATION OF THE TWO MULTI-RESELLING METHODS

while our two methods follow a software-based approach. Using the trusted hardware helps the Nuove system to achieve properties such as preventing an unauthorised reselling, and keeping LI’s overhead at a minimum level. Nonetheless, as mentioned above, using trusted hardware normally imposes an additional cost on both the reseller and buyer.

In the design of our two methods, trusted hardware is not required. Instead, they make use of a contract signing protocol (i.e. RDS protocol described in chapter 4), two digital tokens (RP and MRP), and RPD hosted by the existing license distribution infrastructure. The RDS protocol is used to bind a reseller and a buyer to a reselling process. The RP and MRP are used to indicate that a license is either resalable or multi-resalable. The assistance role played by LI is a simple extension of the role already played by LI in the current DRM systems. For example, the RP/MRP can be embedded in the original license. Payment processing, license activation and revocation are all parts of the functions LI already performs in the existing digital license distribution. Therefore, we can claim that the RRP-MR and HC-MR methods are built on the existing license distribution infrastructure. These methods also extend this infrastructure to support the license multi-reselling facility with the use of cryptographic primitives. Our methods are more cost-effective and provide an alternative to hardware-based solutions.

The RRP-MR and HC-MR methods and the Nuovo system can all protect the integrity of a license. Our two methods make use of LI’s RPD and LI’s signature on a license and its RP or its MRP to achieve this property, while the Nuovo system achieves this property by using a trusted device resident on a reseller’s device.

In our HC-MR method, a buyer is enabled to perform the multi-reselling check on a license being resold. This is achieved by using the verifications \( MrC_1 \) or \( MrC_i \). With the RRP-MR method, the buyer can only verify whether a license is resalable (i.e. by performing the \( RS \) check), he cannot verify how many times this license can be further resold. In the Nuovo system, as an original license is not resalable, this check is not supported.

In our two methods, the monetary interest of resellers and buyers is supported but in the Nuovo system it is not. As described in Chapters 5 and 6, the monetary interest is addressed by making use of the fair and the abuse-free RDS protocol.
which integrates the CS scheme with the existing license distribution infrastructure (i.e., LI). The reseller is able to use and then resell his license at the highest possible price and the buyer can buy a cheap second-hand license at the lowest possible price. This is achieved by the fair and abuse-free RDS protocol.

The Nuovo system allows a consumer to buy rights to resell his license \( N \) times. However, it has not considered the implications of a reseller wanting to resell the original license once he has already resold it \( N \) times. This has been addressed in the design of our RRP-MR and HC-MR methods.

Moreover, in the Nuovo system, LI’s overhead is low as it is only invoked if there is a dispute between a reseller and a buyer. Nonetheless, in our two methods, LI’s overhead is high as it plays an essential role in these methods, although in the HC-MR method, LI’s overhead is better than that in the RRP-MR method.

Furthermore, with the use of trusted hardware, if a reseller has granted to resell \( N \)-copies of his license, the Nuovo method allows the reseller to grant a buyer a license with the right to resell \( M \)-copies, where \( N \geq M \). In our two methods, a reseller can not do so.

With our two methods, a buyer of a second-hand license can detect any cheating committed by a reseller. This is accomplished by using the checks \( RS, LA, \) and \( AKF \) designed for the RRP-MR method and using \( LA_1, LA_i, AKF, MrC_1, \) and \( MrC_i \) designed for the HC-MR method. In the Nuovo system, once a reseller has circumvented his device, he can cheat a buyer, whereas the buyer cannot detect this cheating.

In the Nuovo system and in our method, RRP-MR, a buyer can stop a multi-reselling process of a license but the buyer can not do so with the HC-MR. This is achieved by using the trusted hardware in the Nuovo system and by the RD activation request in the RRP-MR method.

With the use of Nuovo system, a license reselling process requires 6 or 12 exponentiation operations (Exp#) when LI is not used and is used, respectively (see chapter 3). On the other hand, a license reselling process needs 31 Exp# when the RRP-MR method is used and 30 Exp# when the HC-MR method is used. This means the Nuovo system is more efficient than our two methods. Nonetheless, the Nuovo system achieves this efficiency by using a special trusted hardware which usually incurs a charge to resellers and buyers.
6.7 Chapter Summary

This chapter has presented two novel methods, RRP-MR and HC-MR, to support a multi-reselling facility of digital licenses. While achieving this facility, the two methods offer a number of interesting features: (1) they do not require the use of any trusted hardware, thus making our methods more cost-effective; (2) they are secure and fair, i.e. can thwart any attempts of unauthorised resellings, and can prevent resellers from continuing using their resold licenses and they enable buyers to check whether licenses they are about to purchase are resalable and have not already been resold. These features are achieved by making use of (a) the existing license distribution infrastructure (including LI); (b) two novel digital tokens: \( RP \) with the RRP-MR method, and \( MRP \) with the HC-MR method; and (d) the 2M-RDS protocol. LI can help to prevent any unauthorised reselling of a license and to prevent a reseller from using his license after it has been resold. In addition, LI can guarantee a fair reselling between a reseller and a buyer. LI will not send the license to the buyer until LI has received the payment from the buyer. LI also will not forward the buyer’s payment to the reseller until the reseller has revoked the license, thus preventing continued use of the resold license by the reseller. The two methods also benefit from the non-repudiation property of the RDS protocol to prevent a buyer and a reseller from falsely denying having participated in the reselling of a license, thus protecting the content owner’s right. They also make use of the digital tokens, \( RP \) and \( MRP \), to allow buyers to perform the re-saleability and multi-reselling checks, respectively. These checks enable the buyers to detect any unauthorised reselling at an early stage of a reselling process reducing the risk of DoS attack. In addition, the use of \( MRP \) helps the HC-MR method to keep LI’s overhead at a minimum level. Furthermore, comparison with related work has shown that our two method support multi-reselling with fairness and non-repudiation without making use of trusted hardware, thus they are cost-effective. It has also shown that our two methods are the first piece of work that support consumer’s monetary interest, making them more attractive.
Chapter 7

Conclusion and Future Work

The focus of this thesis was on designing secure and fair solutions to enable a digital license reselling in a DRM context. The need for a license reselling facility comes from the fact that current DRM systems do not allow reselling of a digital license which is legitimately bought from a License Issuer (LI) whereas the first-sale doctrine allows a consumer to resell what he/she has legitimately purchased including a digital license.

Due to the nature of a digital content that can easily be copied and redistributed at virtually no cost, current DRM systems only allow a device, to which a license was issued, to use the license and its associated content. If this license is moved to another device(s), it cannot be used to render its associated content. Therefore, to support reselling of a digital license, the DRM systems have to ensure that (1) a license reseller cannot (a) continue to use or to resell a license once he has resold it, and (b) falsely deny having resold his license if he has indeed resold it; (2) a License Issuer should be able to trace a license buyer if he has violated usage rights of the license; (3) a license buyer should be able to (a) use the license to render its corresponding content once he has received it from the reseller, and (b) be able to verify whether a license he is about to purchase is resalable.

A license reselling process could be conducted between a reseller and a buyer who had never met face-to-face beforehand or have no prior business history, i.e. they do not trust each other to behave fairly. While conducting a reselling process over the Internet, one entity (e.g. a buyer) must always initialise the process. The buyer sends his/her item (i.e. a payment) before receiving the expected item (i.e. a license) from the other entity (e.g. a reseller). In this case, upon the receipt of
the payment, the reseller could refuse to send the license, terminate the reselling transaction, and disappear from the Internet. As a result, the buyer will be left in a disadvantageous position. Therefore, a license reselling process should support not only the security of a DRM system but also the fairness of the reselling process.

In addition to the security and fairness issues discussed above, if a license reselling process accommodates market power supporting both reseller’s and buyer’s monetary interests, the process would be much more attractive. A buyer’s interest, in this case, is to pay as little as possible for a second-hand license. A reseller’s interest, on the other hand, is to maximise the price of the license as much as possible.

7.1 Thesis Contributions

The aim of this thesis was to investigate and design solutions to support digital license reselling in the current DRM systems while protecting the underlying security of these systems and providing fairness in a reselling process. Achieving this aim has led to the following novel contributions.

1. The design, formal verification, prototyping, and evaluation of the Reselling Deal Signing (RDS) protocol, a novel fair and abuse-free contract signing protocol that allows a reseller and a buyer to sign a contract called Reselling Deal (RD). The novelty of the protocol lies in the following aspects:

   (a) To the best of the author’s knowledge, the RDS protocol is the first protocol that integrates the Concurrent Signature (CS) scheme with the existing license distribution infrastructure (i.e. LI) to support fair contract signing in a license reselling process. By this integration, the protocol captures the strengths of both the CS scheme and the existing license distribution infrastructure while overcoming their weaknesses. The protocol achieves the properties of strong fairness and abuse-freeness.

   (b) The RDS protocol does not require a dedicated TTP to achieve the fairness property. Rather, it makes use of LI and the CS scheme that minimise the involvement of LI while achieving the desired properties.
7.1. THESIS CONTRIBUTIONS

(c) The informal and the formal analyses of the security properties of the RDS protocol have been conducted. The ATL logic and the model checker MOCHA were used to perform the formal analysis.

(d) The theoretical and prototype-based evaluations have been carried out demonstrating that the RDS protocol is more secure and more efficient than other proposals.

2. The design, analysis, and evaluation of the Reselling Deal (RD) method, a novel method supporting fair and secure digital license reselling. The RD method ensures that a license can only be resold once. This method makes use of (1) a novel Reselling Permission (RP) token (which is attached to a resalable license to differentiate it from an non-resalable license. It also allows a buyer to verify whether a license he is about to buy is resalable before engaging in the license purchase process. This prevents the buyer from experiencing denial of service (DoS) attacks and fraud and forgery attacks by the reseller and other entities), and (2) three protocols, 2M-RDS (2-Message Reselling Deal Signing) protocol, RDA (Reselling Deal Activation) protocol, and RDC (Reselling Deal Completion) protocol. A reseller and a buyer use the 2M-RDS protocol to sign a deal, RD, containing a license price, and the terms and conditions for the license being resold. During the execution of this protocol, the buyer uses the RP token to ensure that the license is resalable and still valid for resale. The buyer then invokes the RDA protocol with the LI who has issued the license. This invocation allows the buyer to make the payment stated in the deal and to receive an activated license from LI. Finally, LI executes the RDC protocol with the reseller to revoke the resold license from the reseller’s device and to forward the payment to the reseller. Upon a successful execution of the RD method, the reseller and the buyer will receive the payment and the activated license, respectively, thus achieving fairness. This fair reselling is achieved while preserving content owners’ rights. That is, the method does not allow a license reseller to (1) continue to use his resold license, (2) resell his license more than once, and (3) resell a non-resalable license.

3. The design, analysis, and evaluation of the Multiple License Reselling (MLR) the method. This novel method allows one license to be resold N times with N different consumers. We have designed two variants of the MLR method,
RRP-MR and HC-MR. The RRP-MR method is designed in favour of buyers. It gives buyers the power to decide whether to continue or to stop a multi-reselling process of a license. This method is a straightforward extension of the RD method. Like the RD method, the RRP-MR method makes use of (1) the novel RP token, RPD (Read-only Public Directory) running by LI, and (2) three protocols, 2M-RDS protocol, RDA protocol, and RDC protocol. With this method, a buyer can repeatedly obtain new reselling permission for a license being resold. Each such permission allows the resale of the license once. In other words, while executing the RDA protocol with LI, the buyer has an option to pay extra fees to LI to get a resalable license. If the buyer does not request a new reselling permission, the multi-reselling process of the license stops. The RRP-MR method achieves the same level of security and fairness as in the case of the RD method.

The HC-MR method offers two extra features over the RRP-MR method, (1) keeping LI’s overhead at a minimum level by not issuing RP at each reselling of a license, and (2) allowing a buyer to verify how many times a license can be further resold. These two features are achieved by utilising a novel MRP token along with an online check of a Read-only Public Directory (RPD) which is updated by LI after each reselling. The MRP is digital token which is generated and signed by LI, and then attached to a multi-resalable license while it is being issued by LI to a consumer. It can be used for N resellings of a license without LI’s further signatures, thus so reducing overhead costs imposed on LI. The MRP token makes use of the hash chain primitive to reduce LI’s involvement in the process. It further enables a buyer to check whether a license is multi-resalable and how many times it can be further resold. The limit on the number of resellings for each license is integrity-protected by LI’s signature which along with the online checking facility allows a buyer to confirm how many more times the corresponding license is allowed to be resold prior to its purchase. The HC-MR method, like both the RD and RRP-MR methods, makes use of the protocols, 2M-RDS, RDA, and RDC to facilitate the multiple resellings, and LI to achieve fair and secure license reselling.

4. The novel methods, RD, RRP-MR, and HC-MR allow content owners to
establish a new business model in the DRM context. In this model, content owners can generate different types of licenses, non-resalable, single-resalable, and multi-resalable licenses. They can set different prices for the different types of licenses. For example, they may set a higher price for multi-resalable licenses, a lower price for single-resalable ones, and the lowest price for non-resalable ones. By reselling a resalable license, the reseller actually acts as an agent for the content owner. This will not only bring monetary benefits to the reseller, but also bring benefits to the content owner. These benefits include increased market penetration and license distributions.

Our solution consisting of the novel methods, RD, RRP-MR, and HC-MR, has the following features. Firstly, it is designed based upon the existing distribution infrastructure (i.e. LI and DRM client). This will give us the benefit that whatever technology is available at LI can easily be made available to a license reselling process. Also, by this approach, the license reselling facility does not require the use of any additional trusted hardware, thus keeping the cost low for consumers who are the most important entity in the value chain. Secondly, it is the first license reselling solution which classifies licenses into three types, non-resalable licenses, single resalable licenses, and multiple resalable licenses, and proposed methods (as summarised above) to support secure and fair single and multiple resellings of resalable licenses. The novel RP token and the novel MRP token are used to indicate that a license is single resalable and multiple resalable, respectively, whereas a license, without any reselling permission, is non-resalable. Thirdly, our solution is the first to support of the market power with license reselling. This market power enables both buyers and resellers to maximise their respective monetary interests. Embedding this idea in the RDS protocol is novel. It allows resellers and buyers to sign as many deals (contracts) as possible with other buyers and resellers, respectively, without violating each other’s rights. To the best of our knowledge, our license reselling solution is the first that provides market power feature in the context of reselling a digital license. Fourthly, our solution is also the first piece of work to support fairness and non-repudiation in a license reselling process. This is achieved using the Reselling Dealing Signing (RDS) protocol by which a reseller cannot falsely deny having resold his license if he has
indeed done so. Fifthly, our solution is the first piece of research work that supports multiple resellings of a digital license (i.e. it allows a license to be resold N times by N different consumers). Sixthly, the solution is the first license reselling proposal that only makes use of a software-based mechanism (i.e. no trusted hardware is required) to revoke a resold license from its reseller’s device. This is achieved by making use of the tamper-proof DRM client installed on the reseller’s device and LI. This license revocation is accomplished during the RDC protocol executed between LI and the reseller. The last, but no the least, our solution has been analysed against its requirements and against potential attacks and threats. The analysis showed that the methods satisfy their respective requirements. In addition, they can thwart potential attacks and threats.

7.2 Directions for Future Work

We introduce the following recommendations as directions for future work:

**Reselling a Domain-license:** The concept of an Authorised Domain (or a domain for short) was first introduced by Digital Video Broadcasting (DVB) organisation [148]. A domain is a group of devices, which are equally authorized to play/view digital contents. An example of such a domain is a home network where a number of devices are interconnected, so a digital content can be moved from device to device seamlessly. The main goal of the domain is to allow a trade off between the interest of consumers and that of content providers. Consumers want to freely access their contents on a number of devices they own whereas content owners would like to ensure that usage rights of their content are protected. FairPlay DRM is an example of DRM systems that have implemented the concept of the authorised domain. It allows a consumer to play/view a content on up to five devices and on an unlimited number of iPods.

In a domain-based DRM, a license (i.e. a domain-license) is issued to a domain identity instead of a device identity. This means that any device belonging to the domain can use the domain-license to access the corresponding content. This concept allows consumers to play/view their contents on more than one device. The main characteristics of the domain-based DRM are [3]:

- A license is bound to a domain when it is acquired;
7.2. DIRECTIONS FOR FUTURE WORK

- A license can be moved among the domain devices;
- A device may be a member of a number of domains.

To support a domain-license reselling, there is a main challenging issue that needs to be addressed. That is, for a domain-license, once it has been resold, it should be revoked from all the devices in the domain. This is a very challenging task as a content owner or a License Issuer needs to first track on which domain device the license was being accessed. LI then revokes or assures the revocation of this license from all the domain devices. This is necessary to ensure that a license reseller can no longer use the license on any domain device after the license has been resold.

**Further reducing LI’s involvement:** In our solution, LI is involved in a payment process carried out between a reseller and a buyer. As described earlier, to get a resold license, a buyer pays LI a payment agreed with a reseller. LI then forwards this payment to the reseller. Involving LI in the payment process places an additional workload on LI. There may be a way by which the buyer can directly pay the reseller without LI’s involvement and without compromising the properties of fairness and abuse-freeness. For example, e-Payment (e.g. e-Cash and e-Checks) method may be used to achieve this.

**Privacy-preserving License Reselling:** Protecting consumers’ privacy is one of the main challenging issues in the digital world. In the context of digital license reselling, an interesting privacy property issue is how to support a reseller and a buyer to conduct a license reselling process in an anonymous/pseudonymous manner without increasing the risk of compromising other security properties. In our solution as discussed in Chapters 6 and 7, a reseller and a buyer have to sign a contract called a Reselling Deal (RD) to conduct a reselling process. The signed deal is then used by LI (1) as evidence that they have engaged in the license reselling process, and (2) to achieve a fair reselling process. Consequently, LI can learn the reseller’s and the buyer’s identities from the signed deal.

Another interesting area for further work could be investigating the use of techniques such as the unlinikability method [51], blind signature [52, 53], blind decryption, hash chain [54], and anonymous cash [55], to address the issue of privacy-preserving of license reselling.

**License Reselling in Cloud DRM:** Recently, a growing number of organisations are moving their contents into a cloud computing environment. Cloud computing is a new trend in information technology and scientific computing.
It is concerned with moving computing and data from desktops and portable PCs into large data centres [149]. DRM could be used as a security technology to protect the contents stored in the cloud. Zou et al. [150] have proposed the first DRM-based cloud to realise this idea. Cloud computing usually involves high commutation and communication costs [151]. If a new service (i.e. license reselling) is to be added to the current DRM-based cloud, how this could be achieved while keeping the commutation and communication costs at a minimal level is an interesting area for future research.

**License Reselling in Mobile DRM:** The number of mobile phones is increasing every day. In addition, mobile phones have become quite powerful devices, which can be used to view video clips and long films. OMA DRM is a standard DRM for mobile phones. It is widely used by many mobile phones, e.g. Nokia, Sony Ericsson, Philips, Motorola, Samsung, and mobile operators. Allowing a license reselling facility in this OMA DRM would make it very attractive to both users and content owners. The users can gain some money by reselling unwanted contents whereas the content owners can establish a new business model supporting the use of both resalable and non-resalable licenses. A main challenge to support license reselling in OMA DRM (in addition to the issues addressed in our solution in this thesis) is how to use lightweight encryption techniques to achieve the required security properties making the solution more suited to power-driven mobiles.

### 7.3 Deployment Requirements

The requirements needed to deploy the proposed methods differ for LI and resellers’/buyers’ view. To adopt these methods, LI needs to support the following: (1) modifying the structure of his licenses such that (a) a single resalable license comes with a Reselling Permission (RP), and (b) a multiple resalable license is attached with a Multiple Reselling Permission (MRP), whereas (c) the non-resalable license remains the same, (2) modifying the issuing process of the license such that LI and a consumer (reseller) securely share the keystone corresponding to the keystone fix contained in RP or MRP, (3) modifying the DRM client by implementing the proposed set of protocols. As the proposed set of protocols are only based on software technology, we argue that it is not difficult for these protocols to be adopt by the current DRM systems. The technologies required for
deploying these protocols are the same technologies being used with the current DRM system, e.g., DRM client and protected communication channels provided by the use of SSL protocol. LI may need to increase the computational capabilities of his servers to meet the additional processing load as caused by license reselling processes.

On the other hand, resellers/buyers do not need any special requirement of hardware or software. Like in the current DRM system, they need to get the DRM client supported by LI to (1) perform the normal DRM operations (e.g., playing, viewing, checking revoked license, etc) and (2) resell/buy a second-hand license operations which should have implemented in the DRM client as discussed above.
Bibliography


Appendix A

All Protocols in Detail

This Appendix gives a description for the protocols proposed in chapter 4 and chapter 5.

A.1 RDS Protocol in Detail

The main cryptographic primitive used in designing the RDS protocol is the CS scheme outlined in Appendix B, Section 3.5.1. Prior to executing the protocol, Alice and Bob first run the SETUP algorithm to agree on parameter values, described in Appendix B, to be used in the signing process. They then engage in executing the RDS protocol, depicted in Figure 4.4. As Alice holds \( f \) and \( ks \) (assumption 2), she will initiate the protocol execution.

**Step 1:** Alice ambiguously signs RD and sends it in \( Msg_1 \) to Bob. Signing RD requires the following operations.

1. Choosing a random number, \( r \in Z_q \); and

2. Creating \( A\text{Sign}_A \). To do this, she runs the ASIGN algorithm, described in Appendix B, with the following inputs: \( PK_A/ SK_A, PK_B, RD \) and \( f \) that is contained in \( RP_{Lic} \) (assumption 2). This is formally described as follows:

\[
A\text{Sign}_A = (s_A, h_A, f) = \text{ASIGN}(PK_A, PK_B, SK_A, f, RD),
\]

where:

(a) \( f = H_1(ks) \)

(b) \( h_A = (h - f) \mod q \); where: \( h = H_2(g^r PK_B^f mod p || RD) \);

(c) \( s_A = (r - h_A SK_A) \mod q \).
Once $A_{Sign_A}$ is generated, Alice constructs and sends Bob $Msg_1$, i.e.

$$Alice \rightarrow Bob: \; Msg_1 = \{RD||A_{Sign_A}||RP_{Lic}\}$$

$RP_{Lic}$ contains the license identity, $Lic$, and the keystone fix, $f$. $RP_{Lic}$ is to assure Bob that $Lic$ is resalable before he signs $RD$ for it. $RP_{Lic}$ also contains $LI$’s signature, $Sig_{LI}(Lic||f)$, to allow Bob to verify that: (1) $f$ is indeed issued by $LI$, and (2) $f$ is bound to $Lic$.

**Step 2:** Once Bob receives $Msg_1$, he performs verification $B_{V1}$ (i.e. he checks whether equation (A.1) holds, where $Lic_{RD}$ and $Lic_{RP_{Lic}}$ are the license ID contained in $RD$ and $RP_{Lic}$, respectively). If $B_{V1}$ is negative, Bob terminates the protocol run. If it is positive, Bob performs $B_{V2}$ shown in Figure A.1. If $B_{V2}$ is positive, Bob performs $B_{V3}$ i.e. checking whether equation (A.2) holds.

$$Lic_{RD} = Lic_{RP_{Lic}}$$  \hspace{1cm} \text{(A.1)}$$

If equation (A.2) holds, Bob creates $A_{Sign_B}$ on $(RD||A_{Sign_A}) = RD_{A_{Sign_A}}$. The purpose for Bob to sign $RD_{A_{Sign_A}}$ instead of signing on the $RD$ directly, is to prevent Alice from abusing $A_{Sign_B}$ once she receives it in $Msg_2$ (i.e. to achieve abuse-freeness). This is because if Bob signs on $RD$, Alice can use $ks$ to make $A_{Sign_B}$ binding to Bob and then show it to a third party. In this case, Alice can gain some advantage over Bob.

To create $A_{Sign_B}$ on $RD_{A_{Sign_A}}$, Bob runs the ASIGN algorithm with the following inputs: $PK_B/SK_B$, $PK_A$, $f$ (the same $f$ used by Alice) and $RD_{A_{Sign_A}}$. This is formally described as follows:

$$A_{Sign_B} = (s_B, h_B, f) = ASIGN(PK_B, PK_A, SK_B, f, RD_{A_{Sign_A}})$$

where:

1. $f = H_1(ks)$

2. $h_B = (h - f) \mod q$; and $h = H_2(g^{r'} PK_A^f \mod p || RD_{A_{Sign_A}})$, and $r'$ is a random number chosen from $Z_q$;

3. $s_B = (r' - h_B SK_B) \mod q$.

Upon generating $A_{Sign_B}$, Bob sends $Msg_2$ to Alice, i.e.

---

1Similar to the case of $B_{V1}$, if any of the verifications, $B_{V2}$, $B_{V3}$, $B_{V4}$, $A_{V1}$, and $B_{V2}$ is negative, the protocol execution will be terminated.
Bob → Alice: $Msg_2 = \{RD_{ASign_A}||ASign_B\}$.

**Step 3:** On the receipt of $Msg_2$, Alice performs $A_V1$. If $A_V1$ is positive, she performs $A_V2$, i.e. confirms that equation (A.3) holds.

$$h_B + f = H_2(g^a PK_B h_B PK_A f \mod p || RD_{ASign_A}) \mod q \quad (A.3)$$

If $A_V2$ is positive, Alice uses $ks$ to convert $ASign_A$ and $ASign_B$ to signatures which are binding to their respective signers. As a result, Alice gets a signed RD which is of the form $(RD, ASign_A, ASign_B, ks)$. In order for Bob to also obtain a signed RD, Alice encrypts $ks$ using $PK_B$, and then sends it in $Msg_3$ to Bob. The keystone fix, $f$, is included in $Msg_3$ to enable Bob to identify that $ks$ is the one corresponding to $f$ used in $Msg_1$ and $Msg_2$.

Alice → Bob: $Msg_3 = \{f||E_{PK_B}(ks)\}$.

Once Bob receives $Msg_3$, he obtains $ks$ and performs $B_V4$ to confirm that the hash value of this $ks$ is equal to $f$ contained in $RP_{lic}$. Bob then uses this $ks$ to convert $ASign_A$ and $ASign_B$ to signatures which are binding to their respective signers. As a result, Bob will obtain a signed RD which is of the form $(RD, ASign_A, ASign_B, ks)$. At this stage, both Alice and Bob have obtained the RD that has been signed by Alice and Bob, thus achieving fair RD signing.
A.2 2M-RDS Protocol in Detail

The main cryptographic primitive used in designing the 2M-RDS protocol is the CS scheme outlined in Appendix B. Prior to executing the 2M-RDS protocol, Alice and Bob negotiate and agree on system parameter values to be used during the protocol run. To establish these parameters, both run the SETUP algorithm described in Appendix B. Upon the execution of this algorithm, both Alice and Bob will generate their respective private parameters, namely their private keys $SK_A$ and $SK_B$. These private keys $SK_A$ and $SK_B$ are chosen uniformly at random from $Z_q$. The private keys are then used to generate the corresponding public keys $PK_A$ and $PK_B$, as follows: $PK_A = g^{SK_A}(mod\ p)$ and $PK_B = g^{SK_B}(mod\ p)$.

Alice and Bob should keep their respective private keys secret and only exchange their public keys with each other. In addition, the execution of the SETUP algorithm will generate the following items: two large prime numbers, $p$ and $q$, such that $q| (p - 1)$, a generator $g$ of a multiplicative subgroup of order $q$ in $Z_p^*$, and two cryptographic hash functions, $H_1$ and $H_2$: $\{0, 1\}^* \rightarrow Z_q$. $H_1$ is only used in creating a hash value of the keystone $ks$ while $H_2$ is used to compute other hash values for signature generation. Also the public parameters include descriptions of message space $M$, signature space $S$, keystone space $K$, and keystone fix space $F$. These spaces are defined as follows: $S \equiv F = Z_q$ and $M \equiv K = \{0, 1\}^*$.

Once the execution of the SETUP algorithm is completed, Alice and Bob will engage in executing the 2M-RDS protocol, depicted in Figure 5.3. As Alice holds the keystone, $ks$, (see the assumptions), she will initiate the protocol execution.

**Step 1:** As illustrated in Figure A.2, Alice ambiguously signs RD and sends $Msg_{1RDS}$, to Bob. Signing RD requires the following operations.

![Figure A.2: $ASign_A$ signature generation on RD](image)

1. Choose a random number, $r \in Z_q$; and
2. Create $A_{Sign}$. To do this, she runs the ASIGN algorithm, described in Section ??, with the following inputs: $PK_A$, $SK_A$, $PK_B$, RD and $f$ given in $RP_{Lic}$ (assumption 2). This is formally described as follows:

$$A_{Sign} = (s_A, h_A, f) = \text{ASIGN}(PK_A, PK_B, SK_A, f, RD),$$

where:

(a) $f = H_1(ks)$
(b) $h_A = h - f \mod q$;

where: $h = H_2(g^r P K_{B}^f \mod p||RD)$;
(c) $s_A = r - h_A S K_A(\mod q)$.

Once $A_{Sign}A$ is generated, Alice sends Bob $Msg_{1_{RDS}}$, i.e.

$Alice \rightarrow Bob: \{RD||A_{Sign}A||RP_{Lic}\}$,

where: $RP_{Lic} = \{Lic||f||RP_{period}||Sig_{LI}(Lic||f||RP_{period})\}$

**Step 2**: Once Bob receives $Msg_{1_{RDS}}$, he performs the following verifications.

**Verification $B_{V1}$**

In this verification, as shown in Figure A.3, Bob checks the correctness of LI’s signature on each of $RP_{Lic}$ and $Lic-File$. If $B_{V1}$ is negative, Bob terminates the protocol run. If it is positive, it means that $RP_{lic}$ and $LicFile$ are indeed issued by LI and not tampered with. In other words, Bob can verify that the usage rights granted in the license file are identical to those he has negotiated.

![Figure A.3: Verification $B_{V1}$](image)

**Verification $B_{V2}$**

In this verification, Bob checks whether the equality (A.4) holds, where $Lic_{RD}$ is the license identity, $Lic$, contained in RD, $Lic_{RP_{Lic}}$ is the license identity given
in $RP_{Lic}$, and $Lic_{Lic−File}$ is the license identity specified in the license file. With this verification, Bob confirms that the license negotiated for RD is identical to the one provided in the received $RP_{Lic}$. This says that the license has been authorised for resale by LI.

$$Lic_{RP_{Lic}} = Lic_{RD} = Lic_{Lic−File}$$  \hspace{1cm} (A.4)

**Verification $B_{V3}$**

In this verification, Bob checks whether the equality (A.5) holds, where $f_{RP_{Lic}}$ is the keystone fix provided in $RP_{Lic}$ and $f_{ASign_A}$ is the one used in generating $ASign_A$. $B_{V3}$ assures Bob that $f$ is the right keystone fix to be used in the signing process.

$$f_{RP_{Lic}} = f_{ASign_A}$$  \hspace{1cm} (A.5)

**Verification $B_{V4}$**

In this verification, Bob checks that $RP_{Lic}$ has not already been used in a previous reselling of $Lic$. This is to prevent unauthorised multiple resellings of $Lic$. This is done by having Bob check LI’s website to verify that $f$, contained in $RP_{Lic}$, is not on the resold license list published on this website. This verification is discussed in detail in Section 5.4.2.

**Verification $B_{V5}$**

As illustrated in Figure A.4, in this verification, Bob checks the correctness of $ASign_A$. This is done by checking whether equation (A.6) holds.

$$h_A + f = H_2(g^{s_A} PK_A P K_B f_{mod p} || RD) mod q$$  \hspace{1cm} (A.6)

![Figure A.4: Verification of $B_{V5}$](image)

If any of these verifications is negative, Bob aborts the protocol run. If they are
all positive, as depicted in Figure A.5. Bob creates $ASign_B$ on $(RD||ASign_A) = RD_{ASign_A}$. The purpose of Bob to signing $RD_{ASign_A}$, instead of signing on the RD directly, is to prevent Alice from abusing $ASign_B$ once she receives it in $Msg_{2RDS}$. This is because if Bob signs on RD, Alice can use $ks$ to make $ASign_B$ binding to Bob and then show it to a third party. In this case, Alice may gain some advantage over Bob. For example, Alice may use Bob’s signature on RD to coerce Carol to sign a better deal, $RD_2$, for Lic. However, if $ASign_A$ is a part of $ASign_B$, when $ASign_B$ becomes binding to Bob, $ASign_A$ will be binding to Alice as well. In this way, it would be difficult for Alice to benefit by showing a third party the RD that is already signed by both Alice and Bob, thus achieving abuse-freeness.

![Figure A.5: $ASign_B$ signature generation on $(RD||ASign_A)$](image)

To create $ASign_B$ on $RD_{ASign_A}$, Bob runs the ASIGN algorithm with the following inputs: $PK_B, SK_B, PK_A, f$ (the same $f$ used by Alice) and $RD_{ASign_A}$. $ASign_B$ is of the following form:

$$ASign_B = (s_B, h_B, f) = ASIGN(PK_B, PK_A, SK_B, f, RD_{ASign_A}),$$

where,

1. $f = H_1(ks)$;
2. $h_B = h - f \mod q$, and $h = H_2(g^r PK_A^f \mod p||RD_{ASign_A})$, and $r'$ is a random number chosen from $Z_q$;
3. $s_B = r' - h_B SK_B \mod q$.

Once $ASign_B$ is generated, Bob sends $Msg_{2RDS}$ to Alice, i.e.

$Bob\rightarrow Alice: Msg_{2RDS} = \{RD_{ASign_A}||ASign_B\}$.

**Step 3**: Once $Msg_{2RDS}$ is received, Alice performs verifications $A_{V1}$ and $A_{V1}$.

**Verification $A_{V1}$**

In $A_{V1}$, Alice checks if equality (A.7) holds, where $f_{ASign_A}$ is the keystone fix used in generating $ASign_A$ and $f_{ASign_B}$ is the keystone fix used in generating
ASign\textsubscript{B}. This is to confirm to Alice that in generating ASign\textsubscript{B}, Bob has used the same \( f \) Alice has used in generating ASign\textsubscript{A}.

\[ f_{\text{ASign}_A} = f_{\text{ASign}_B} \quad \text{(A.7)} \]

**Verification \( A_{V2} \)**

In this verification, as shown in Figure A.6, Alice confirms that the equality (A.8) holds.

\[
h_B + f = H_2(g^a PK_B^{h_B} PK_A^{f_A} (mod p)||RD||\text{ASign}_A)(mod q)) \quad \text{(A.8)}
\]

If either \( A_{V1} \) or \( A_{V2} \) is negative, Alice aborts the protocol run. Otherwise, if both are correct, Alice considers that she has signed a deal, RD, with Bob. Alice may then choose one of the following options: (1) to check LI’s website to find out whether Bob has paid LI for her license, Lic; and (2) to opt for another deal, RD\textsubscript{2}, i.e. to sign another RD with a different buyer, as Bob may have also accepted another deal, RD\textsubscript{3} with another license reseller. In other words, both Alice and Bob can sign as many deals as they like with other entities. Both can then choose the best one to suit their respective interests. If Bob wants to proceed with this deal, RD, he will execute the RDA protocol to be discussed in Section 5.4.2.

### A.3 RDA Protocol in Detail

The RDA protocol details are summarized in Figure 5.4. To activate the \( RD_{\text{Pre-official}} \) deal, Bob generates an \( RD_{\text{activation request}} \). This request consists of:
• **RD\textsubscript{Pre-official}:** This is a reselling deal that has been ambiguously signed by both Alice and Bob. In other words, it is the outcome of the execution of the 2M-RDS protocol and its structure is shown as Figure 5.5.

• **Payment\textsubscript{B}:** This is the cost of the license, Lic that should be paid by Bob to LI before Bob can activate Lic. This value should be equal to the license price specified by both Alice and Bob in the deal, RD. As stated in assumption 1, it is assumed that there is a payment system by which Bob can make this payment to LI;

• (RP\textsubscript{Lic}): This is the Reselling Permission for Lic which has been received by Bob during the execution of the 2M-RDS protocol. This RP\textsubscript{Lic} is of the form: \( RP\textsubscript{Lic} = \{Lic || f || RP\textsubscript{period} || Sig\textsubscript{LI}(Lic || f || RP\textsubscript{period})\} \);

• **Sig\textsubscript{B}:** This is Bob’s signature on RP\textsubscript{Lic} and Payment\textsubscript{B}.

Once Bob has generated the RD activation request, he sends it to LI in \( Msg1\textsubscript{RDA} \).

\textbf{Bob} \rightarrow \textbf{LI}: \( Msg1\textsubscript{RDA} = \{RD\textsubscript{Pre-official} || Payment\textsubscript{B} || RP\textsubscript{Lic} || Sig\textsubscript{B}(Payment\textsubscript{B} || RP\textsubscript{Lic})\} \)

When LI receives \( Msg1\textsubscript{RDA} \), LI performs verifications, \( LI\textsubscript{V1} \) through to \( LI\textsubscript{V6} \) described below. If all these verifications are positive, LI activates the deal, RD; otherwise, LI rejects the activation request.

**LI’s verifications:**

\textbf{LI\textsubscript{V1}:} Through this verification, LI confirms that equality (A.9) holds, where Payment\textsubscript{B\_RDA} is the value of Payment\textsubscript{B} provided in \( Msg1\textsubscript{RDA} \); Payment\textsubscript{B\_RD} is the value of Payment\textsubscript{B} stated in the received RD; and Payment\textsubscript{B\_LI} is the value of Payment\textsubscript{B} made to LI. If LI\textsubscript{V1} is positive, LI proceeds to LI\textsubscript{V2}. Otherwise, LI terminates the process.

\[
\text{Payment}_{B\_RDA} = \text{Payment}_{B\_RD} = \text{Payment}_{B\_LI} \quad \text{(A.9)}
\]

\textbf{LI\textsubscript{V2}:} This is for LI to confirm that Bob has submitted the deal, RD, before the deadline, \( RD\textsubscript{DLB} \), has expired. This check enables LI to be assured that both Alice and Bob have agreed to activate the deal, RD, within the specified deadline. Before \( RD\textsubscript{DLB} \), Bob should have sent LI an activation
APPENDIX A. ALL PROTOCOLS IN DETAIL

request, and before $RD_{DL_A}$, Alice should have confirmed to LI that she has revoked the license, $Lic$. If LI receives the RD activation request after the deadline, $RD_{DL_B}$, LI will reject the request. Otherwise, LI proceeds to perform $LI_{V_3}$.

$LI_{V_3}$: LI checks the correctness of $Sig_B(Payment_B||RP_{Lic})$. As illustrated in Figure A.7, in this verification, Bob verifies Bob’s signature on the RD activation request.

![Figure A.7: Verification $LI_{V_3}$](image)

$LI_{V_4}$: LI checks the legitimacy of $Lic$. This check consists of four further checks: $LI_{V_{4.1}}$, $LI_{V_{4.2}}$, $LI_{V_{4.3}}$, and $LI_{V_{4.4}}$.

$LI_{V_{4.1}}$: (Double Reselling Check) In this check, LI ascertains that the license, $Lic$, has not already been resold. This is done by using a Read-only Public Directory (RPD). As illustrated in Figure A.8, the RPD consists of three fields: license identity, and keystone and keystone fix assigned to this license identity. For example, once a license, $Lic$, has been resold using $RP_{Lic}$, LI publicises $ks$ and $f$ corresponding to $RP_{Lic}$ on this RPD. While performing $LI_{V_{4.1}}$, if LI finds $ks$, corresponding to $RP_{Lic}$, has been published in the field, “Used keystone” of RDP, this means that $Lic$ has already been resold. Consequently, LI will reject this RD activation request. Otherwise, LI proceeds to perform $LI_{V_{4.2}}$. 
By performing LI\textsubscript{V4.1}, the license, Lic, is prevented from being resold multiple times. In addition, any buyer can use this RPD to check whether \( RP_{Lic} \) has already been used in reselling Lic.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{RPD.png}
\caption{RPD hosted by LI}
\end{figure}

LI\textsubscript{V4.2}: (\( RP_{Lic} \) Validity) LI checks that the inequality (A.10) holds, where \( RecDate_{RP} \) is the date and time at which LI has received \( RP_{Lic} \), and \( EnDate_{RP} \) is the date and time by which \( RP_{Lic} \) will expire. If this inequality does not hold, then \( RP_{Lic} \) has expired and LI rejects the activation request. If it holds, then \( RP_{Lic} \) is still valid to be used in reselling Lic and LI proceeds to perform LI\textsubscript{V4.3}.

\begin{equation}
RecDate_{RP} \leq EnDate_{RP}
\end{equation}

LI\textsubscript{V4.3}: (\( RP_{Lic} \) authenticity) LI verifies whether \( RP_{Lic} \) has been altered. This is done, as illustrated in Figure [A.3], by verifying LI’s signature on \( RP_{Lic} \), i.e. verifying \( \text{Sig}_{LI}(Lic||f||RP_{period}) \). If this signature is invalid, the license, Lic, is deemed as non-resalable and LI sends a negative acknowledgement to Bob in \( Msg_{2RDA} \). Thus, performing LI\textsubscript{V4.3} helps LI to prevent reselling a non-resalable license. On the other hand, if LI\textsubscript{V4.3} is positive, the license, Lic, is considered as resalable. LI then proceeds to perform LI\textsubscript{V4.4}.

LI\textsubscript{V4.4} (License Identity Check) In this check, LI verifies that the equality (A.11) holds, where Lic\textsubscript{RD} is the license identity stated in RD, and
$\text{Lic}_{RP_{\text{lic}}}$ is the license identity specified in $RP_{\text{lic}}$.

$$\text{Lic}_{RD} = LI_{RP_{\text{lic}}} \quad \text{(A.11)}$$

If this equality does not hold, LI will stop the reselling process. This check prevents a reseller or a buyer from replacing $RP_{\text{lic}}$ with another perhaps less valuable reselling permission. If $LI_{V,4}$ is positive, LI then proceeds to perform $LI_{V,5}$.

$LI_{V,5}$: (Reseller’s Signature Verification) LI verifies whether $\text{ASign}_A = (s_A, h_A, f)$ attached to $RD_{\text{pre-official}}$ is indeed created by Alice. This is done by verifying whether equality (A.12) holds.

$$h_A + f = H_2(g^{s_A} PK_A^{h_A} PK_B^f mod p || RD) mod q \quad \text{(A.12)}$$

If equality (A.12) does not hold, Alice is not the true originator of $\text{ASign}_A$. If equality (A.12) holds, and if $f = H_1(ks)$, then $\text{ASign}_A$ is indeed created by Alice.

$LI_{V,6}$: (Buyer’s Signature Verification) LI verifies whether $\text{ASign}_B = (s_B, h_B, f)$ attached to $RD_{\text{pre-official}}$ is indeed created by Bob. That is, LI checks whether equality (A.13) holds:

$$h_B + f = H_2(g^{s_B} PK_A^{h_B} PK_B^f mod p || (RD || \text{ASign}_A) mod q) \quad \text{(A.13)}$$

If equality (A.13) does not hold, Bob is not the true originator of $\text{ASign}_B$. If equality (A.13) holds, and if $f = H_1(ks)$, it means that $\text{ASign}_B$ is created by Bob.

From the discussion above, the following remarks can be drawn. Firstly, $LI_{V,1}$ and $LI_{V,2}$ prevent Bob from activating RD without making an adequate payment before a pre-defined deadline, $RD_{DL_B}$. Also, performing $LI_{V,1}$ at this early stage of RD activation reduces the risk of Denial of Service (DoS) attack on LI. This is because before the required payment is received, no further verifications will be performed. In other words, if an inadequate payment is received, LI will not need to do signature verifications specified in $LI_{V,3}$, $LI_{V,4}$, $LI_{V,5}$, and $LI_{V,5}$ which are computationally more expensive than $LI_{V,1}$ and $LI_{V,2}$. This design consideration
can reduce unnecessary computational load on LI, and reduce the chance of LI becoming a target of DoS attack.

Secondly, LI_{V2} and LI_{V4} are necessary in order to protect the content owners’ rights. With LI_{V2}, LI will be assured that both Alice and Bob are responsible for activating the deal, RD, within the specified deadlines: \( RD_{DL_A} \) and \( RD_{DL_B} \). Thus, as discussed in Section 5.1.3.3, neither Alice nor Bob can later falsely deny that they are unaware of this responsibility. LI_{V4} ensures LI that the license, Lic, is legitimate. In other words, LI_{V4} enables LI to prevent (1) the reselling of a non-resalable license, (2) double reselling of a resalable license.

Thirdly, LI_{V5} and LI_{V6} are to make sure that both Alice and Bob have indeed committed to the RD before the reselling is allowed to proceed. This can also help to prevent any potential dispute between Alice and Bob as both entities must have signed the given RD before the reselling process can be successfully completed. LI_{V5} also protects LI from any repudiation attack from Alice. In other words, if Alice has indeed resold her license, Lic, but falsely denies doing so, LI can use her signature on RD as evidence to prove that she has definitely resold the license, Lic.

If all the verifications, LI_{V1} through to LI_{V6}, are positive, LI accepts RD and performs the following three tasks before sending lic_{Activated} to Bob in Msg2RDA.

1. LI re-issues the license, Lic, to become the activated license, lic_{Activated}. In this way, when it is installed on Bob’s device, Bob can access it on his device. In the license re-issuance, LI does not issue another reselling permission to lic_{Activated}, thus making it non-resalable. The purpose of this license re-issuance is for LI to assign a new ID to lic_{Activated}, thus solving the license ID problem (see Section 5.1.3) if the Global-LRL is used.

2. LI binds the license, lic_{Activated}, to Bob’s device to address the DRM license transfer problem (see Section 2.3). This license binding is achieved by encrypting the license, lic_{Activated}, using the public key of Bob’s DRM client. In this way, only Bob’s DRM client can use its private key to decrypt lic_{Activated} and use it. This license binding enables LI to trace Bob should Bob violate any of the usage rights specified in the license.

3. LI marks Alice’s license, Lic, as resold and publishes the license details on LI’s website. This is done, as shown in Figure A.8, by LI running a Read-Only Public Directory (RPD) containing resold licenses and their
corresponding keystones and keystone fixes. LI can use this RPD to perform the verification, $\text{LI}_{V41}$.

Once LI has performed all the above tasks, LI, in $\text{Msg2}_{RDA}$, sends the license, $\text{Lic}_{\text{Activated}}$, encrypted by Bob’s public key, to Bob, thus achieving fairness for Bob.

$$\text{LI} \rightarrow \text{Bob}: \text{Msg2}_{RDA} = \{\text{Lic}_{\text{Activated}}||\text{Sig}_{\text{LI}}(\text{Lic}_{\text{Activated}})\}_\text{PK}_B$$

After Bob receives $\text{Msg2}_{RDA}$, his DRM client uses its private key to decrypt $\text{Msg2}_{RDA}$ to get the license, $\text{Lic}_{\text{Activated}}$. He then performs Verification $B_{V6}$ to verify that LI has indeed created this message. Now, Bob is able to access the license, $\text{Lic}_{\text{Activated}}$, on his device.

### A.4 RDC Protocol in Detail

The RDC protocol is executed to assure LI that Alice’s resold license, $\text{Lic}$, has indeed been revoked and to transfer payment for the license reselling to the reseller, Alice. After the successful execution of the RDA protocol, as described in A.3, LI performs two tasks (1) prepares an LRL-update containing the ID of $\text{Lic}$ and posts this on LI’s website; (2) publishes on LI’s website that $\text{Lic}$ has been resold. Once this LRL-update is published on LI’s website, it is Alice’s responsibility to regularly check LI’s website to see whether her license has been resold. If so, Alice has to obtain the LRL-update, and install it on her device. She then executes the RDC protocol with LI. If the deadline, $\text{RD}_{DL_A}$, expires and Alice does not do so, she can neither claim for the payment, $\text{Payment}_B$, nor resell her license again.

When Alice obtains the LRL-update which is of the form

$$\{\text{LRL}_{\text{update}}||\text{Sig}_{\text{LI}}(\text{LRL}_{\text{update}})\}$$

Alice’s DRM client performs the verification, $A_{V3}$.

**Verification $A_{V3}$:**

In $A_{V3}$, Alice’s DRM client checks the authenticity of the downloaded LRL-update. That is, as shown in Figure A.9, $\text{Sig}_{\text{LI}}(\text{LRL}_{\text{update}})$ will be verified. If $A_{V3}$ is negative, the DRM client will not allow this LRL-update to be installed. If it is positive, the LRL-update will be installed on Alice’s device to update the LRL already installed on her DRM client.

---

2Note that, in this case, LRL-update is delivered to Alice using the pull mode described in Section 5.1.3.2. The push mode can also be used to deliver the LRL-update. That is, once LI successfully executes the RDA protocol with Bob and prepares an LRL-update, he can send it to Alice who installs it on her device and then launches the RDC protocol.
Upon successful installation of the LRL-update on Alice’s device, Alice’s DRM client generates and sends LI $\text{Msg1}_{\text{RDC}}$. This $\text{Msg1}_{\text{RDC}}$ is to confirm that an up-to-date LRL has been installed on Alice’s DRM client (i.e. Alice’s resold license is revoked).

**Alice’s DRM client $\rightarrow$ LI: $\text{Msg1}_{\text{RDC}} =$**

$$\{A_{PK}||\text{InstallTime}||LRL\text{Info}||\text{Sig}_A(A_{PK}||\text{InstallTime}||LRL\text{Info})\},$$

where:

- $A_{PK}$: This is Alice’s DRM client ID.

- $LRL\text{Info}$: This is information about two LRLs, (1) the LRL-update being installed, and (2) the old LRL which was already installed on Alice’s device. $LRL\text{Info}$ contains the following fields:

  1. $LRL\text{Version}$: This is the latest version of the LRL being installed on Alice’s device.

  2. $ID_{LRL\text{-update}}$: This is the identity of the LRL-update that is published on LI’s website and that should be installed on Alice’s device;

  3. $LRL\text{Install-Status}$: This is either $success$ or $fail$. This is to show whether the LRL-update has been successfully installed or not.

- $\text{InstallTime}$: This is the time at which Alice’s license has been revoked. In other words, it is the time when the LRL-update is installed on Alice’s device.
This time is securely provided by Alice’s DRM client, and it is synchronized with LI’s clock (See assumption 12).

$Sig_A$: This is the signature of Alice’s DRM client which is created on the $Msg_{1,RDC}$.

Once LI receives $Msg_{1,RDC}$ from Alice, LI performs the following verifications to confirm if Alice’s license has indeed been revoked.

Verification LI$_{V7}$

In this verification, LI checks the correctness of Alice’s DRM client, $Sig_A$, on $Msg_{1,RDC}$. If LI$_{V7}$ is negative, LI may ask Alice to re-execute the RDC protocol. Of course, this re-execution should be done before the deadline, $RD_{DLA}$. If Alice’s DRM client signature is valid, LI goes to LI$_{V8}$.

Verification LI$_{V8}$

In LI$_{V8}$, LI confirms that the equality (A.14) holds, where $LRC_{LRL,Version}$ is $LRL_{Version}$ installed on Alice’s device and provided in the LRC message, i.e. $Msg_{1,RDC}$, and $LI_{LRL,Version}$ is the latest version of LRL published on LI’s website.

\[ LRC_{LRL,Version} = LI_{LRL,Version} \quad (A.14) \]

Verification LI$_{V9}$

In LI$_{V9}$, LI checks that the equality (A.15) holds, where $LRC_{ID_{LRL,update}}$ is $ID_{LRL,update}$ installed on Alice’s device and provided in the LRC message, i.e. $Msg_1$, and $LI_{ID_{LRL,update}}$ is $ID_{LRL,update}$ of the latest update of LRL published on LI’s website after reselling Lic. Note that $ID_{LRL,update}$ is very important for LI as it assures LI that Alice has installed the current version of the LRL-update published on LI’s website after reselling the license, Lic.

\[ LRC_{ID_{LRL,update}} = LI_{ID_{LRL,update}} \quad (A.15) \]

If all the above verifications are positive, LI is assured that Alice cannot use her resold license, Lic, any more. LI can then forward Alice the payment received from Bob during the execution of the RDA protocol. This could be done by having LI send a cheque with this payment value to Bob. LI can inform Alice of this payment by sending $Msg_{2,RDC}$. That is,

\[ LI \rightarrow Alice: \; Msg_{2,RDC} = \{ Payment_B || N_{LI} || Sig_{LI}(Payment_B || N_{LI}) \} \]

After Alice receives $Msg_{2,RDC}$, she performs Verification, $A_{V4}$. This verification checks that the message is authentic, i.e. it is indeed generated by LI. If
$A_{V4}$ is positive, Alice should receive the payment stated in $Msg_2^{RDC}$. This will complete the reselling of the license, $Lic$. 
Appendix B

Cryptographic and Protection Technologies Used in DRM

B.1 Chapter Introduction

DRM Systems make use of different protection technologies (i.e. building blocks) to protect content owners’ rights, to secure digital contents, and to secure content distributions. Typically, symmetric encryption, e.g. Advanced Encryption Standard (AES), is used to protect the contents. Asymmetric encryption is also used to bind a license to a particular consumer (device). Digital certificates with digital signatures are used to authenticate involved entities to each other. PKI is utilised to manage and provide services related to digital certificates. In addition, digital signatures and hash functions are used to protect the integrity of both contents and licenses. Digital watermarking and fingerprinting are used to embed ownership information into contents and to trace copyright violations if a system is successfully attacked, and its content made public. Individualisation is used to make an instance of DRM client unique among all other clients. Tamper resistance technologies are utilized to ensure that the entire DRM system is trusted, and fraud can be prevented. To make sure that a digital content remains confidential and it is correctly received, a secure communication channel is established using SSL.

This Chapter gives an overview of the building blocks mentioned above. In Section B.2 and Section B.3, we discuss the concepts of symmetric and asymmetric encryptions. The hash function and digital signature will be presented in Section B.4 and Section B.5 respectively. Public Key Infrastructure (PKI)
will be introduced in Section B.6. Individualisation and tamper resistance will be discussed in Section B.7 and Section B.8, respectively. Finally, Section B.9 presents the Secure Socket Layer/Transport Layer Security (SSL/TLS) protocol.

## B.2 Symmetric Cryptography

Symmetric encryption (also known as secret-key encryption, or one-key encryption) is an encryption technique in which the same secret key is used for both encryption and decryption. Figure B.1 illustrates an example of the symmetric encryption process.

Symmetric encryption can be used to achieve security of message confidentiality and message authenticity. Since a sender and a receiver are the only entities that know the encryption/decryption key, message confidentiality is preserved. Message authenticity is preserved due to the fact that the key is only known to the two entities.

According to the way in which the plaintext is processed, symmetric encryption algorithms can be further categorised into two classes, stream and block ciphers [152]. In the stream cipher, the plaintext is processed in a bit-wise or byte-wise manner, whereas in the block cipher, the plaintext is broken up into groups of bits (called blocks) of a fixed length and then one block is processed at a time [152].

Well known symmetric encryption algorithms include the Data Encryption Standard (DES [32]), and the Advanced Encryption Standard (AES) [33].

## B.3 Asymmetric Cryptography

Asymmetric cryptography (also referred to as public-key cryptography) is another type of the key-based encryption algorithms. Unlike the symmetric cryptography, the asymmetric cryptography makes use of a pair of keys for encryption and decryption. A key called public key is used to encrypt data, and another key called private key is used to decrypt the data. The public key can be publicised to anyone while the private key must be kept secret and known only to its owner. There is a mathematical relation between the public key and the private key in a pair so that data encrypted with either key can only be decrypted using the other.
As illustrated in Figure B.2, in order for a sender to send a confidential message to a receiver, the sender uses the receiver’s public key to encrypt the message and then sends it to the receiver. To decrypt the encrypted message, the receiver must use his private key. Since this receiver alone is in possession of his private key, only he can decrypt the message. By this way, message confidentiality can be achieved.

Asymmetric encryption does not suffer from the key distribution problem as in the case of symmetric encryption. However, asymmetric encryption is much slower than symmetric encryption (up to 100 times in software and up to 10,000 times in hardware [153]). Therefore, asymmetric encryption is normally not used for bulk message encryption. It is typically combined with symmetric encryption to make use of the advantages of both encryptions. It is mainly used at the beginning of a communication session to exchange a shared key (called session key) between two entities. These two entities will then use this key for subsequent message integrity and confidentiality protection using symmetric encryption. Examples of well known Internet security protocols that make use of this approach are SSL/TLS [95, 112] and IPSec [154].
B.4 Hash Functions

A hash function, $H()$, is a function that takes an arbitrary block of data ($M$) and produces an output of a fixed-size string (known as a hash value, $h$). Any change to the data $M$ will result in a change in the hash value, $h$. A typical cryptographic hash function should satisfy the following requirements [152]:

- The input of $H()$ can be of any length,
- The output of $H()$ has a fixed length,
- $H(x)$ is relatively easy to compute for any given input, $x$,
- $H(x)$ is a one-way function. A hash function $H()$ is said to be one-way if it is computationally infeasible to compute $x$ when $H(x)$ is given.
• $H(x)$ should be collision-free. This means that for a given message $x$, it is computationally infeasible to find a message $y \neq x$ such that $H(x) = H(y)$.

• A strongly collision-free hash function, $H()$, is one for which it is computationally infeasible to find any pair $(x, y)$ such that if $H(x) = H(y)$.

The cryptographic hash function plays an important role in the creation of digital signatures. A hash function can be used to generate a message digest from an original message before a sender signs the message digest with his/her private key. Examples of well-known hash functions are MD5 [155], SHA [156], and SHA-512 [157].

B.5 Digital Signatures

A digital signature is an alternative for a hand-written signature. Typically, a digital signature scheme comprises two main components [152]:

• A signature generation algorithm: This is to generate a digital signature.

• A signature verification algorithm: This is to verify a digital signature.

To digitally sign a message $M$, a sender first applies a hash function $H()$ to the message to generate a digest of the message (hash value), $h$. The sender then encrypts this digest with his private key to generate a digital signature. He then appends this signature to the original message and sends both to a receiver.

To verify the sender’s signature, the receiver first decrypts the signature with the sender’s public key to obtain the message digest, $h$. The receiver then generates another fresh message digest, $h'$, from the received message. If the two messages’ digest are equal, i.e. $h = h'$, then the signature is considered valid. Otherwise, the signature is not valid. Figure B.3 illustrates the process of both signature generation and signature verification.

Digital signatures are very important for the security services. They provide the properties of message integrity, non-repudiation, and message/entity authentication.

• Message integrity: Once a receiver of a signed message has successfully verified the signature, he will be assured that neither the message nor the signature has been modified in transit. Thus, a digital signature is used to verify the integrity of a signed message.
APPENDIX B. BUILDING BLOCKS USED IN DRM

Figure B.3: Digital signature generation and verification

- **Non-repudiation**: A signer of a message cannot later deny having signed the message. As explained above, a signer uses his private key to generate his signature on the message. As the signature is verifiable with the corresponding public-key, so the signature represents undeniable evidence for any signed messages.

- **Authentication**: Digital signatures provide origin authentication for signed messages. This is due to the fact that (1) messages are signed using the private key of the senders; (2) the senders are the only entities that are in possession of these private keys. Therefore, the signed messages should have originated from the claimed senders.

Well-known examples of digital signature schemes, which have been proposed in the literature, include RSA [158], DSA (Digital Signature Algorithm, [159]), El Gamal [160], and Schnorr [161]. Recently, a new type of digital signature scheme, called Concurrent Signature (CS) scheme has been proposed in [8] and [91]. The main difference between the CS scheme and other digital signature schemes is that the CS scheme allows two signers to exchange their respective signatures concurrently. In the following section, we describe in more detail the generation and verification process of the CS scheme. The scheme has been used in the design of our contract signing protocol.
B.6 Public-key Infrastructure (PKI)

Asymmetric cryptosystems are associated with an important security infrastructure, known as Public key Infrastructure (PKI). PKI is an infrastructure which concerns with managing public keys certificates (digital certificates, see Section B.6.1) of all involved entities in an asymmetric cryptosystem. In other words, PKI exists to ensure how other entities can ascertain that a public key is authentic, i.e. it indeed belongs to a particular entity. PKI involves a number of Trusted Third Party (TTP) to manage and verify the identities of the entities wishing to engage in a secure communication through the issuance of digital certificates for these entities [152].

In a typical PKI, a TTP called a registration authority verifies an entity’s identity and then instructs another TTP called a Certificate Authority (CA) to generate and sign a digital certificate for the entity. This certificate contains the public key of this entity which can later use this certificate to prove its identity and establish secure transactions with other entities [152]. Generally, PKI provides a number of services such as issuing, managing digital certificates, and maintaining Certificate Revocation Lists (CRLs).

B.6.1 Digital Certificate

A digital certificate is a data structure which is issued and signed by a Certificate Authority (CA) [152]. As shown in Figure B.4, a digital certificate contains a public key, the purpose of use, e.g. encryption and/or signing, the owner of the public key, the issuer, date of issuing, validity period, signature algorithm used to sign the certificate, and optional extension fields that may be used to customise certificates. The fields specified in a standard certificate are defined by the X.509 Certificate Specification [162].

One of the main applications of digital certificates is electronic commerce. In an E-commerce system, an owner of a secure site will maintain a digital certificate that is checked by a web browser for a secure session. In this case, CA issuing this certificate is asserting that the public key of the owner is authentic. To establish the secure session, the owner and web browser will authenticate each other using their public keys and then create a session key which is used to establish an encrypted session. With this way, when personal details, like credit card numbers, are in transit over the Internet, other entities cannot see them.
Digital Certificate Structure
According to the X.509 standard [162], the structure of a digital certificate is as follows:

- **Version Number**: It is the version of X.509 to which the certificate is issued (at the time of writing the current X.509 version is 3).

- **Serial Number**: It is a unique number to identify the specific certificate issued by a given CA.

- **Issuing CA**: It is the name of the certificate authority which has issued the certificate.

- **CA Digital Signature**: It is the digital signature of the issuing CA.

- **Subject/Owner**: It specifies the owner of the certificate. It may be a person, company, department, network device, or application etc.

- **Owner’s Public Key**: It specifies the public key contained in the certificate and corresponding to the owner’s private key.

- **Validity Period**: It defines two dates during which the certificate is considered valid.

- **Certificate Usage**: It specifies the approved uses of the certificate (e.g. signature and/or encryption).

- **Signature Algorithm**: This is the field containing the identifier of the cryptographic algorithm used by CA to sign the certificate.

- **Extensions**: They allow addition of customised data to digital certificates.

B.6.2 Certification Authority (CA)

A certificate authority (CA) is a TTP which is responsible for validating the identity of a person or organization requesting a certificate[^1]. Upon the verification of the identity, it issues a certificate containing the subject’s public key. The certificate is then digitally signed with the CA’s private key [152].

[^1]: In some scenarios, CA delegates the identity’s validation to another TTP, called a Registration Authority.
B.6.3 Certificate Lifecycles and Key Management

A certificate lifecycle includes (but not necessarily all) the following stages [152]:

- **Key Generation**: It is the stage of creating a public/private key pair to be associated with the certificate.

- **Identify Submission**: It is the stage where an entity’s credentials (e.g. entity’s ID, address, email address) has to be submitted to the CA.

- **Registration**: It is the stage where the identity of the requesting entity is verified and registered by the CA.

- **Certification**: In this stage, the identity of a requesting entity is validated, and a certificate is issued, i.e. generated and digitally signed, by the CA.

- **Distribution**: It is the stage where certificates are published by the CA.

- **Usage**: It is the stage during which a certificate is used by the requesting entity.

- **Expiration**: It is the stage where the certificate becomes expired unless it is renewed or revoked.
APPENDIX B. BUILDING BLOCKS USED IN DRM

- **Revocation**: It is the stage where a certificate is revoked before it expires. For example, if the private key of a certificate is compromised, the corresponding certificate has to be revoked.

- **Suspension**: In this stage a certificate is temporarily suspended. For example, if a certificate holder goes on sabbatical leave and is not planning on using his certificate during this period.

### B.7 Individualization

Individualization is a process by which a DRM client becomes unique from all other DRM clients. With the individualization, each license is tightly issued to a unique DRM client (i.e. playback device). Thus, a license issued and stored in a given device is difficult to move or to be used by another device. The individualization, in DRM context, helps to prevent software hacking on a large scale. It also decreases the potential to revoke all DRM clients if one device is compromised [163, 26].

On the other hand, the individualization introduces a great concern regarding rights portability. For example, a consumer usually wants to play/view her film at her friend’s home or play/view it on her portable devices (iPad, mobile phones, etc.). In this case, to play/view the film on each device, she needs to acquire new licenses for every device. To address this problem, as described in Chapter 2, FairPlay DRM system allows consumers to play/view their content on up to five devices.

As used in Microsoft DRM, a DRM client can be individualised as follows. A DRM system collects consumer’s hardware information or consumer’s identification (e.g. consumer’s public key). The DRM system uses this information to issue a public/private key pair to the consumer device. The private key is then stored by the DRM client, and the public key is used to identify this DRM client. At the time of issuing a license, a License issuer (LI) will use this key to encrypt the license. To use this license, the DRM client will use the private key to decrypt the license [163, 26].
B.8 Tamper resistance

Tamper resistance is a technology that aims to protect trusted software running on a non-trustworthy host. In the DRM context, consumers cannot be considered as trustworthy. Therefore, content owners need to protect their usage rules contained in digital licenses against violations, malicious alterations, etc. Furthermore, content owners may want to ascertain that their rights in the digital license are properly enforced [40].

To prevent a malicious consumer from tampering with the rights entitlement functions of the DRM-enabled applications, it is important to apply a tamper resistance technology. This technology makes hacking extremely difficult and ensures that the DRM client can be trusted to perform operations as designed [26].

There are two types of tamper resistance technologies, software approach and hardware approach. Software approach makes use of software mechanisms to enable tamper resistance. Code obfuscation [164, 165] and self-modifying [166] are two examples of software tamper resistance technology. Code obfuscation transforms a piece of software into a functionally equivalent form which is difficult to understand and analyse. For example, a piece of software may be encrypted and executed in the encrypted form. Self-modifying code is a code which generates a different code while it is being run. This can prevent consumers (hackers) from viewing, understanding, accessing, and modifying the software.

Hardware approach relies on trusted hardware to provide tamper resistance. In this hardware, code is protected from external software attacks. In DRM context, operations such as authentication, rights rendering, and content decryption are executed only in this trusted hardware. Examples of hardware tamper resistant mechanism are the Trusted Platform Module (TPM) specifications [167] and smart card technology which have been used in decoding the signals of satellite TV to prevent subscription fraud [168].

B.9 SSL/TLS Protocol

The Secure Socket Layer (SSL) protocol [95] is one of the most crucial solutions over the Internet. It provides the required level of authentication, confidentiality, and message integrity for the exchanged data. SSL is a transport layer security
APPENDIX B. BUILDING BLOCKS USED IN DRM

protocol which is enveloped by Netscape Communications. Recently, another version of SSL protocol called Transport Layer Security (TLS v1) protocol [112] is developed based on SSL v3. TLS is proposed as Internet Standard from the Internet Engineering Task Force (IETF). Since the design of the TLS v1, the term SSL applies to both protocols SSL and TLS.

Currently, SSL is built into Web browsers and servers, and widely adopted by the current e-commerce community to provide secure communications and transfer of sensitive data (e.g. monetary transactions and credit card numbers). Any URL that requires an SSL communication uses https:// instead of http://.

The SSL protocol comprises two phases. In the first phase, the SSL execution starts with a protocol known as handshake in which communicating entities are mutually authenticated using their public-key certificates and digital signatures. The entities then jointly compute a key called a session key. In the second phase, messages are exchanged between the entities. The session key is used to encrypt the messages to protect their confidentiality.

B.10 Chapter Summary

This chapter has introduced the cryptographic primitives which form the basis for the design of our fair and secure solution to the digital license reselling problem. It has also given an overview of the protection technologies used in the design of DRM systems. To advance the state-of-the-art, the next chapter investigates the existing license reselling systems and analyses their advantages and limitations.
Appendix C

Formal Verification Code: RDS Protocol

This Appendix gives the formal verification details of the RDS protocol. Section C.1 presents the protocol model in the guarded command language. This model is used for model-checking with Mocha. Section C.2 then gives a representation of the security properties as ATL formulae that are used as inputs to Mocha.

C.1 The RDS Protocol Model

………………..Modelling an Honest Alice…………………..

module Alice
/* Variables that could be accessed by Alice and other players*/
external
Msg2,
RD_Act_Req6,
Positive_Result_A : bool
/* Variables controlled by Alice */
interface
RD,
RP_Lic,
E_ks,
Assign_A,
A_v1,
A_v2,
SendMsg1,
SendMsg$,
Stop_A,
Assign_Ba,
Assign_Ba_LI,
Time_Out_A: bool
atom Alice_Sign
controls
RD,
RP_Lic,
E_ks,
Assign_A,
A_v1,
A_v2,
SendMsg1,
SendMsg3,  
Stop_A,  
Asign_Ba,  
Asign_Ba_LI,  
Time_Out_A  
reads RD,  
RD,  
RP_Lic,  
E_ks,  
Asign_A,  
A_v1,  
A_v2,  
SendMsg1,  
SendMsg3,  
Stop_A,  
Asign_Ba,  
Msg2, Asign_Ba,  
RD_Act_Req6,  
Positive_Result_A,  
Asign_Ba_LI,  
Time_Out_A  

/* this is the initialization state for the honest Alice*/  
init  
[ ] true ->  
RD' := true; RP_Lic' := true; Asign_A' := true;  
E_ks' := false; A_v1' := false; A_v2' := false;  
SendMsg1' := false; SendMsg3' := false; Stop_A' := false;  
Asign_Ba' := false; Asign_Ba_LI' := false;  
Time_Out_A' := false  

update  
/* Alice sends Msg1 to Bob*/  
[ ] RD & RP_Lic & Asign_A & "Stop_A & SendMsg1 -> SendMsg1' := true  
/* Alice has received Msg2, i.e. she will receive Asign_Ba*/  
[ ] Msg2 & "Stop_A & Asign_Ba -> Asign_Ba' := true; Time_Out_A' := true  
/* Once Alice has received Msg2, she perform the verification A_v1 and A_v2*/  
[ ] SendMsg1 & Msg2 & Asign_Ba & "Stop_A & A_v1 -> A_v1' := true  
[ ] SendMsg1 & Msg2 & Asign_Ba & "Stop_A & A_v1 & A_v2 -> A_v2' := true  
/* If the verifications A_v1 and A_v2 are positive, Alice encrypts the ks with Bob*/  
[ ] SendMsg1 & Msg2 & Asign_Ba & "Stop_A & A_v1 & A_v2 & E_ks -> E_ks' := true  
/* Once Alice has encrypted the ks, she sends it to Bob*/  
[ ] SendMsg1 & Msg2 & Asign_Ba & "Stop_A & A_v1 & A_v2 & E_ks & SendMsg3 -> SendMsg3' := true  
/* In case Bob has not responded to Alice by Msg3 and he has sent LI RD_Act_Req6  
(an valid RD activation request), Alice will receive a Positive_Result_A (i.e Asign_Ba)  
from LI so achieving fairness for Alice*/  
[ ] RD_Act_Req6 & "Stop_A & Positive_Result_A & Asign_Ba_LI -> Asign_Ba_LI' := true  
[ ] "Stop_A -> Stop_A' := true  
[ ] "Stop_A ->  
endatom  
endmodule  

***************Modelling a Dishonest Alice***************  
module Dis_Alice  
/* Variables that could be accessed by Alice_Dis and other players*/  
external Msg2 : bool  
/* Variables controlled by Alice*/  
interface ks_A,  
RD,  
RP_Lic,  
E_ks,  
Asign_A,  
A_v1,  
A_v2,  
SendMsg1,  
SendMsg3,
C.1. THE RDS PROTOCOL MODEL

Stop_A,
Asign_Ba,
Dis_Alice_Prove2TP,
Time_Out_A: bool

atom AliceSign2
controls ks_A,
RD,
RP_Lic,
E_ks,
Asign_A,
A_v1,
A_v2,
SendMsg1,
SendMsg3,
Stop_A,
Asign_Ba,
Dis_Alice_Prove2TP,
Time_Out_A

reads ks_A,
RD,
RP_Lic,
E_ks,
Asign_A,
A_v1,
A_v2,
SendMsg1,
SendMsg3,
Stop_A,
Asign_Ba,
Msg2,
Asign_Ba,
Dis_Alice_Prove2TP,
Time_Out_A

init
[][] true → ks_A := true;
RD' := true;
RP_Lic' := true;
Asign_A' := true;
E_ks' := false;
A_v1' := false;
A_v2' := false;
SendMsg1' := false;
SendMsg3' := false;
Stop_A' := false;
Asign_Ba' := false;
Dis_Alice_Prove2TP' := false;
Time_Out_A' := false

update
/* Alice sends Msg1 to Bob */
[][] RD & RP_Lic & Asign_A & Stop_A & SendMsg1 → SendMsg1' := true
/* Alice has received Msg2 from Bob, i.e. she will receive Asign_Ba and she will be informed that she needs to send Msg3 within an amount of time i.e. Time_out_A */
[][] Msg2 & Stop_A & Asign_Ba → Asign_Ba' := true; Time_Out_A' := true
/* Once Alice has received Msg2, she perform the verification A-v1 and A-v2 */
[][] SendMsg1 & Msg2 & Asign_Ba & Stop_A & A_v1 → A_v1' := true
[][] SendMsg1 & Msg2 & Asign_Ba & Stop_A & A_v1 & A_v2 → A_v2' := true
/* If Alice wants to prove to a Third Party (TP) that Bob has indeed involved in the RDS. She will use the following predicates */
[][] Asign_Ba & Stop_A & A_v1 & A_v2 & ks_A → Dis_Alice_Prove2TP' := true
/* If the verifications A-v1 and A-v2 are positive, Alice premurly stop the RDS protocol, i.e. Alice neither encycrepts ks with Bob’s key nor sends it to Bob. Alice may do this as she holds the keystone ks, so she can use it to make Bob committed to the RD and she is not */
[][] Stop_A → Stop_A' := true
[][] Stop_A →
endatom

endmodule

/*************** Modelling an Honest Bob ***************/
module Bob

/ Variables that could be accessed by Bob and other players */
external
Msg1,
Msg3,
RD_Act_Req6,
Positive_Result1 : bool

/ Variables controlled by Bob */
interface
RD_b,
RP_Lic_b, Eks_b, Asign_Ab, ks, Eks_b
these are the variable Bob has received in Msg1 and Msg3*

atom BobSign
controls RD_b,
RP_Lic_b, Eks_b,
Asign_Ab,
B_v1,
B_v2,
B_v3,
SendMsg2,
Stop_B,
Asign_B,
Payment_B,
ks,
SendRD_Act_Req6,
ks_L1,
OtherData_B,
Time_Out: bool

reads RD_b,
RP_Lic_b,
Eks_b,
Asign_Ab,
B_v1,
B_v2,
B_v3,
SendMsg2,
Stop_B,
Asign_B,
Payment_B,
Msg1,
Msg3,
ks,
SendRD_Act_Req6,
RD_Act_Req6,
Positive_Result1,
ks_L1,
OtherData_B,
Time_Out

--awaits Time_Out
C.1. THE RDS PROTOCOL MODEL

init
[ true ->
  RD\_b' := false;
  RP\_Lic\_b' := false;
  Eks\_b' := false;
  Asign\_Ab' := false;
  B\_v1' := false;
  B\_v2' := false;
  B\_v3' := false;
  SendMsg2' := false;
  Stop\_B' := false;
  Asign\_B' := false;
  Payment\_B' := false;
  ks' := false;
  SendRD\_Act\_Req6' := false;
  ks' := false;
  ks\_LI' := false;
  OtherData\_B' := false;
  Time\_Out' := false
]

update
/* Bob receives Msg1 from Alice, he will receive Asign\_A and RP\_Lic */
[ /
  Msg1 & 'Asign\_Ab & 'Stop\_B -> RD\_b' := true ; RP\_Lic\_b' := true ; Asign\_Ab' := 
  true
/
  Once Bob has received Msg1, he performs the verification B\_v1 and B\_v2 */
[ /
  Msg1 & 'Stop\_B & B\_v1 -> B\_v1' := true
/
  Msg1 & 'Stop\_B & B\_v2 -> B\_v2' := true
/
  If the verifications B\_v1 and B\_v2 are positive, Bob generates Asign\_B */
[ /
  Msg1 & 'Stop\_B & B\_v1 & B\_v2 & 'Asign\_B -> Asign\_B' := true
/
  Once Bob has generated the Asign\_B, he sends it to Alice */
[ /
  Msg1 & 'Stop\_B & B\_v1 & B\_v2 & Asign\_B & 'SendMsg2 -> SendMsg2' := true
/
  Bob receives Msg3 from Alice, he will receive the keystone ks to make 
  Asign\_A bindig to Alice */
[ /
  SendMsg2 & Msg3 & 'Stop\_B & 'Eks\_b -> Eks\_b' := true
/
  Once Bob has received Msg3, he performs the verification B\_v3 */
[ /
  SendMsg2 & Msg3 & 'Stop\_B & Eks\_b & B\_v3 -> B\_v3' := true
/
  If B\_v3 is true, Bob will get the keystones ks */
[ /
  SendMsg2 & Msg3 & 'Stop\_B & Eks\_b & B\_v3 & 'ks -> ks' := true
/
  Bob may send LI an request when the offer is very good, so he does not 
  reply to Alice by Mes2 but directly send LI an completed RD activation 
  request */
[ /
  Msg1 & Asign\_Ab & RD\_b & RP\_Lic\_b & 'Stop\_B & B\_v1 & B\_v2 & Asign\_B & 
  Payment\_B & 'SendRD\_Act\_Req6 -> SendRD\_Act\_Req6' := true
/
  Or Bob has sent Msg2 to Alice but have not received Msg3 within the 
  time\_out, so he sends LI an D activation request, SendRD\_Act\_Req6 = [
  Msg1+Msg2+Payment\_B] */
[ /
  'Time\_Out -> Time\_Out' := true
/
  Msg1 & Asign\_Ab & RD\_b & RP\_Lic\_b & 'Stop\_B & B\_v1 & B\_v2 & Asign\_B & 
  Time\_Out & Payment\_B & 'SendRD\_Act\_Req6 -> SendRD\_Act\_Req6' := true
/
  In case of sending RD\_Act\_Req6, Bob will receive a positive result from LI 
  (i.e. the keystone ks and otherdate (the license)) */
[ /
  SendRD\_Act\_Req6 & RD\_Act\_Req6 & 'Stop\_B & Positive\_Result1 & 'ks\_LI -> ks\_LI 
  ' := true;
  OtherData\_B' := true;
  'Stop\_B -> Stop\_B' := true
/
  'Stop\_B ->

endatom

endmodule
APPENDIX C. FORMAL VERIFICATION CODE: RDS PROTOCOL

```
RD_Act_Req5,
Negative_Result1,
Negative_Result2,
Negative_Result3,
Negative_Result4,
Negative_Result5 : bool

/* Variables controlled by Alice */
/**RD−b, RP−Lic−b, Eks−b, Asign−Ab, these are the variable Bob has received in Msg1*/
interface
RD_b,
RP_Lic_b,
Eks_b,
Asign_Ab,
B_v1,
B_v2,
B_v3,
SendMsg2,
Stop_B,
Asign_B,
Payment_B,
SendRD_Act_Req1,
SendRD_Act_Req2,
SendRD_Act_Req3,
SendRD_Act_Req4,
SendRD_Act_Req5,
k_s_L1,
k_s,
Dis_Bob_Prove2TP : bool

atom BobSign2
controls
RD_b,
RP_Lic_b,
Eks_b,
Asign_Ab,
B_v1,
B_v2,
B_v3,
SendMsg2,
Stop_B,
Asign_B,
Payment_B,
SendRD_Act_Req1,
SendRD_Act_Req2,
SendRD_Act_Req3,
SendRD_Act_Req4,
SendRD_Act_Req5,
k_s_L1,
k_s,
Dis_Bob_Prove2TP
reads
RD_b,
RP_Lic_b,
Eks_b,
Asign_Ab,
B_v1,
B_v2,
B_v3,
SendMsg2,
Stop_B,
Asign_B,
Payment_B,
Msg1,
Negative_Result1,
Negative_Result2,
Negative_Result3,
Negative_Result4,
Negative_Result5,
SendRD_Act_Req1,
SendRD_Act_Req2,
SendRD_Act_Req3,
SendRD_Act_Req4,
SendRD_Act_Req5,
```
C.1. THE RDS PROTOCOL MODEL

SendRD_Act_Req4,
SendRD_Act_Req5,
RD_Act_Req1,
RD_Act_Req2,
RD_Act_Req3,
RD_Act_Req4,
RD_Act_Req5,
ks_LI,
k,
Dis_Bob_Prove2TP

init
[] true ->
  RD_b' := false;
  RP_Lic_b' := false;
  Eks_b' := false;
  Asign_Ab' := false;
  B_v1' := false;
  B_v3' := false;
  SendMsg2' := false;
  Stop_B' := false;
  Asign_B' := false;
  Payment_B' := true;
  SendRD_Act_Req1' := false;
  SendRD_Act_Req2' := false;
  SendRD_Act_Req3' := false;
  SendRD_Act_Req5' := false;
  ks_LI' := false;
k := false;
Dis_Bob_Prove2TP' := false

update

/* Bob receives Msg1 from Alice, he will receive Asign_A, RP_Lic and RD */
[] Msg1 & "Asign_A" & "Stop_B" -> RD_b' := true ; RP_Lic_b' := true ; Asign_Ab' := true
[] Also Once Bob has received Msg1, he may not reply to Alice by Mes2 but send LI an incompeled RD activation request*/
[] SendRD_Act_Req1' := true

/* Bob may also send LI another incompeled request*/
[] Msg1 & "Stop_B" & "B_v1" -> B_v1' := true
[] SendRD_Act_Req1 & RD_Act_Req1 & "Stop_B" & "SendRD_Act_Req1" -> SendRD_Act_Req1' := true

/* Bob may also send LI a third an incompeled request*/
[] Msg1 & "Stop_B" & "B_v1" & "B_v2" -> B_v2' := true
[] SendRD_Act_Req3 := true

/* if the verifications B_v2 is positive, Bob generates Asign_B */
[] SendRD_Act_Req3 := true

/* if the verifications B_v2 is positive and Asign_B is generated, Bob sends SendRD_Act_Req3 to LI */
[] Msg1 & "Stop_B" & "B_v1" & "B_v2" & Asign_B & "SendRD_Act_Req3" -> SendRD_Act_Req3' := true

/* Bob may also send LI a fourth an incompeled request.*/
[] Msg1 & "Stop_B" & "B_v1" & "B_v2" & Asign_B & Payment_B & "SendRD_Act_Req4" -> SendRD_Act_Req4' := true

/* Bob may also send LI a fifth an incompeled request.*/
[] Msg1 & "Stop_B" & "B_v1" & "B_v2" & Asign_B & "SendRD_Act_Req5" -> SendRD_Act_Req5' := true

/* Bob receives Negative_Result from LI Bob request does not pass LI's verifications*/
[] SendRD_Act_Req1 & RD_Act_Req1 & "Stop_B" & Negative_Result1 & "ks_LI" -> ks_LI' := false ; true
[] SendRD_Act_Req2 & RD_Act_Req2 & "Stop_B" & Negative_Result2 & "ks_LI" -> ks_LI' := false ; true
[] SendRD_Act_Req3 & RD_Act_Req3 & "Stop_B" & Negative_Result3 & "ks_LI" -> ks_LI' := false ; true
[] SendRD_Act_Req4 & RD_Act_Req4 & "Stop_B" & Negative_Result4 & "ks_LI" -> ks_LI' := false ; true
[] SendRD_Act_Req5 & RD_Act_Req5 & "Stop_B" & Negative_Result5 & "ks_LI" -> ks_LI' := false ; true

"Stop_B" -> Stop_B' := true
"Stop_B" ->
endatom
endmodule

/*****Modelling the License Issue (LI) which evaluate the RD submited by Bob
*****
module LI
/*/ Variables that could be accessed by LI and other players*/
external
  RD_Act_Req1,
  RD_Act_Req2,
  RD_Act_Req3,
  RD_Act_Req4,
  RD_Act_Req5,
  RD_Act_Req6 : bool
/*/ Variables controlled by Alice */
interface
  LI_v1,
  LI_v2,
  LI_v3,
  LI_v4,
  LI_v5,
  RD_L,
  Payment_B_1,
  RP_Lic_L,
  Assign_A_1,
  Assign_B_1,
  RD_Act_Req1,
  RD_Act_Req2,
  RD_Act_Req3,
  RD_Act_Req4,
  RD_Act_Req5,
  RD_Act_Req6,
  SendNegative_Result1,
  SendNegative_Result2,
  SendNegative_Result3,
  SendNegative_Result4,
  SendNegative_Result5,
  SendPositive_Result_B,
  SendPositive_Result_A : bool

atom LIchecks
controls
  LI_v1,
  LI_v2,
  LI_v3,
  LI_v4,
  LI_v5,
  RD_L,
  Payment_B_1,
  RP_Lic_L,
  Assign_A_1,
  Assign_B_1,
  SendNegative_Result1,
  SendNegative_Result2,
  SendNegative_Result3,
  SendNegative_Result4,
  SendNegative_Result5,
  SendPositive_Result_B,
  SendPositive_Result_A
reads
  LI_v1,
  LI_v2,
  LI_v3,
  LI_v4,
  LI_v5,
  RD_L,
  Payment_B_1,
  RP_Lic_L,
  Assign_A_1,
  Assign_B_1,
init

[ ] true →
  LIv1 := false;
  LIv2 := false;
  LIv3 := false;
  LIv4 := false;
  LIv5 := false;
  RDl := false;
  PaymentB := false;
  RPlic := false;
  AsignA := false;
  AsignB := false;
  SendNegativeResult1 := false;
  SendNegativeResult2 := false;
  SendNegativeResult3 := false;
  SendNegativeResult4 := false;
  SendNegativeResult5 := false;
  SendPositiveResultB := false;
  SendPositiveResultA := false

update

/* LI receives RD_Act_Req1 = [Msg1] from Bob */
  [ ] RD_Act_Req1 & ~RP_llic := true; AsignA := true; RDl := true

/* LI receives RD_Act_Req6 = [Msg1 + Msg2+ PaymentB] from Bob (complete RD) */
  [ ] RD_Act_Req6 & "AsignB & ~RP_llic := true; AsignA := true;
      RDl := true; PaymentB := true; RP_llic := false

/* RD_Act_Req6 is passed all LI’s verifications will be positive, LI sends Bob a Positive_Result_B (i.e Ks) and also sends Positive_Result_A to Alice (i.e. Asign_B) */
  [ ] RD_Act_Req6 & LIv1 & LIv2 & LIv3 & LIv4 & LIv5 & "SendPositiveResultB := true;
      SendPositiveResultA := true

/* LI will not verified, so LI sends Bob Negative_Results */
  [ ] RD_Act_Req1 & "PaymentB := false
  [ ] RD_Act_Req1 & "LIv1 & SendNegativeResult1 := SendNegativeResult1 := true

/* LI receives RD_Act_Req2 = [Msg1 + PaymentB] from Bob and LIv1 will be true */
  [ ] RD_Act_Req2 & "RP_llic := true; AsignA := true; RDl := true;
      PaymentB := true
  [ ] RD_Act_Req2 & "LIv1 & SendNegativeResult2 := SendNegativeResult2 := true

/* As RD_Act_Req2 is not complete, LI sends Bob a Negative_Results as LIv1 is negatives */
  [ ] RD_Act_Req2 & "AsignB := false
  [ ] RD_Act_Req2 & "LIv4 & SendNegativeResult2 := SendNegativeResult2 := true

/* when RD_Act_Req3, LIv1 will not be true as RD_Act_Req3 does not contain Payment */
  [ ] RD_Act_Req3 & "AsignB := true; RDl := true
  [ ] RD_Act_Req3 & "AsignB & RDl := LIv5 := true

/* Since RD_Act_Req has no Msg1, LI sends Bob a Negative_Results as LIv1 is negatives */
  [ ] RD_Act_Req3 & "PaymentB := false
APPENDIX C. FORMAL VERIFICATION CODE: RDS PROTOCOL

```plaintext
[] RD_Act Req3 & ~LI v1 & SendNegative_Result3 -> SendNegative_Result3 := true
/* As Payment_B is in RD_Act Req4 LI v1 will be true */
[] RD_Act Req4 & ~Assign_B l -> Assign_B l := true; RD_L := true; Payment_B l := true
[] Since RD_Act Req4 has no Msg1, LI also sends Bob a Negative_Results (LI v1 is negative)
[] RD_Act Req4 & ~Assign_A l -> LI v4 := false
[] RD_Act Req4 & ~LI v4 & SendNegative_Result4 -> SendNegative_Result4 := true
/* As RD_Act Req5 does not contain payment, LI v1 is negative, no ks */
[] RD_Act Req5 & ~Assign_B l & ~RP_Lic l -> Assign_B l := true; Assign_A l := true; RD_L := true; RP_Lic l := true
[] RD_Act Req5 & Assign_B l & Assign_A l & RD_L & RP_Lic l -> LI v2 := true; LI v3 := true; LI v4 := true; LI v5 := true
/* Since RD_Act Req5 does not contain payment, LI also sends Bob a Negative_Results */
[] RD_Act Req5 & ~Payment_B l -> LI v1 := false
[] RD_Act Req5 & ~LI v1 & ~SendNegative_Result5 -> SendNegative_Result5 := true
endatom
endmodule

/* Modelling the operational communication channel between Bob and LI */
module OperationalChann_B_LI
/* Variables that could be accessed by operationalChann—AB and other players */
external
SendRD_Act Req1, SendRD_Act Req2, SendRD_Act Req3, SendRD_Act Req4, SendRD_Act Req5, SendRD_Act Req6, SendNegative_Result1, SendNegative_Result2, SendNegative_Result3, SendNegative_Result4, SendNegative_Result5, SendPositive_Result_B, SendPositive_Result_A : bool

/* Variables controlled by OperationalChann_B_LI */
interface
RD_Act Req1, RD_Act Req2, RD_Act Req3, RD_Act Req4, RD_Act Req5, RD_Act Req6, Negative_Result1, Negative_Result2, Negative_Result3, Negative_Result4, Negative_Result5, Positive_Result_B, Positive_Result_A : bool

atom Channed_B_LI
contROLS
RD_Act Req1, RD_Act Req2, RD_Act Req3, RD_Act Req4, RD_Act Req5, RD_Act Req6, Negative_Result1, Negative_Result2, Negative_Result3, Negative_Result4,
```

C.1. THE RDS PROTOCOL MODEL

```
Negative_Result5, Positive_Result_B
reads
RD_Act_Req1, RD_Act_Req2, RD_Act_Req3, RD_Act_Req4, RD_Act_Req5, RD_Act_Req6,
SendRD_Act_Req1, SendRD_Act_Req2, SendRD_Act_Req3, SendRD_Act_Req4, SendRD_Act_Req5, SendRD_Act_Req6,
SendNegative_Result1, SendNegative_Result2, SendNegative_Result3, SendNegative_Result4, SendNegative_Result5,
SendPositive_Result_B, SendPositive_Result_A, Negative_Result1, Negative_Result2, Negative_Result3, Negative_Result4, Negative_Result5,
Positive_Result_B
init
||true ->
| RD_Act_Req1' := false;
| RD_Act_Req2' := false;
| RD_Act_Req3' := false;
| RD_Act_Req4' := false;
| RD_Act_Req5' := false;
| Negative_Result1' := false;
| Negative_Result2' := false;
| Negative_Result3' := false;
| Negative_Result4' := false;
| Negative_Result5' := false;
| Positive_Result_B' := false
update
| SendRD_Act_Req1 & 'RD_Act_Req1 -> RD_Act_Req1' := true
| SendRD_Act_Req2 & 'RD_Act_Req2 -> RD_Act_Req2' := true
| SendRD_Act_Req3 & 'RD_Act_Req3 -> RD_Act_Req3' := true
| SendRD_Act_Req4 & 'RD_Act_Req4 -> RD_Act_Req4' := true
| SendRD_Act_Req5 & 'RD_Act_Req5 -> RD_Act_Req5' := true
| SendRD_Act_Req6 & 'RD_Act_Req6 -> RD_Act_Req6' := true
| SendNegative_Result1 & 'Negative_Result1 -> Negative_Result1' := true
| SendNegative_Result2 & 'Negative_Result2 -> Negative_Result2' := true
| SendNegative_Result3 & 'Negative_Result3 -> Negative_Result3' := true
| SendNegative_Result4 & 'Negative_Result4 -> Negative_Result4' := true
| SendNegative_Result5 & 'Negative_Result5 -> Negative_Result5' := true
| SendPositive_Result_B & 'Positive_Result_B -> Positive_Result_B' := true
endatom
endmodule

/*Modelling the operational communication channel between Alice and LI*/
module OperationalChann_A LI

/* Variables that could be accessed by operationalChann_A LI and other players*/
external

SendPositive_Result_A: bool

/* Variables controlled by OperationalChann_A LI*/
interface

Positive_Result_A : bool

atom Channel_A LI

controls
```
APPENDIX C. FORMAL VERIFICATION CODE: RDS PROTOCOL

Positive_Result_A
reads
SendPositive_Result_A,
Positive_Result_A
init
[] true -> Positive_Result_A' := false
update
[] SendPositive_Result_A & ~Positive_Result_A -> Positive_Result_A' := true
endatom
endmodule

/* Modelling the unreliable communication channel between Alice and Bob */
module UnreliableChann_AB
/* Variables that could be accessed by the unreliableChann-AB and other players*/
external
SendMsg1,
SendMsg2,
SendMsg3 : bool

/* Variables controlled by Alice */
interface
Msg1,
Msg2,
Msg3: bool

atom ChannedAB
controls
Msg1,
Msg2,
Msg3
reads
Msg1,
Msg2,
Msg3,
SendMsg1,
SendMsg2,
SendMsg3
init
[] true ->
   Msg1' := false;
   Msg2' := false;
   Msg3' := false
update
[] SendMsg1 & ~Msg1 -> Msg1' := true
[] SendMsg2 & ~Msg2 -> Msg2' := true
[] SendMsg3 & ~Msg3 -> Msg3' := true
true ->
   idle action
endatom
endmodule

/* Modelling the operational communication channel between Alice and Bob*/
module OperationalChann_AB
/* Variables that could be accessed by perationalChann-AB and other players*/
external
SendMsg1,
SendMsg2,
SendMsg3 : bool

/* Variables controlled by Alice */
interface
Msg1,
Msg2,
Msg3: bool
atom ChannedAB
controls
Msg1,
Msg2,
C.2. ATL Formulae Used in The Verification of The RDS Protocol

```
C.2. ATL FORMULES USED IN THE VERIFICATION OF THE RDS PROTOCOL

Msg3 reads
Msg1, Msg2, Msg3, SendMsg1, SendMsg2, SendMsg3

init [] true ->

Msg1' := false;
Msg2' := false;
Msg3' := false

update

SendMsg1 & ~Msg1 -> Msg1' := true
SendMsg2 & ~Msg2 -> Msg2' := true
SendMsg3 & ~Msg3 -> Msg3' := true

endatom
endmodule

/* Combinations of players used in the execution of the RDS protocol analysis*/
RDS_Alice_Bob_UnreliableCh_AB := Alice || Bob || LI || UnreliableChann_AB
RDS_Alice_Bob_LI_UnreliableCh_AB := Alice || Bob || LI || UnreliableChann_AB || OperationalChann_B_LI || OperationalChann_A_LI
RDS_Dis_Alice_Bob_LI_UnreliableCh_AB := Dis_Alice || Bob || LI || UnreliableChann_AB || OperationalChann_B_LI || OperationalChann_A_LI
RDS_Alice_Dis_Bob_LI_UnreliableCh_AB := Alice || Dis_Bob || LI || OperationalChann_AB || OperationalChann_B_LI || OperationalChann_A_LI
RDS_Dis_Alice_Bob_OperationalCh_AB := Dis_Alice || Bob || LI || OperationalChann_AB || OperationalChann_B_LI
RDS_Dis_Alice_Bob_UnreliableCh_AB := Dis_Alice || Bob || LI || UnreliableChann_AB || OperationalChann_B_LI
RDS_Dis_Bob_Alice_UnreliableCh_AB := Dis_Bob || Alice || LI || UnreliableChann_AB || OperationalChann_A_LI || OperationalChann_B_LI

C.2 ATL Formulae Used in The Verification of The RDS Protocol

atl "Fairness for Bob"
"<<Dis_Alice,UnreliableChann_AB,OperationalChann_B_LI>>
F (Asign_Ba & "<<Bob>> F(Asign_Ab & (ks | ks_LI))));

atl "Fairness for Alice"
"<<Dis_Bob,UnreliableChann_AB,OperationalChann_B_LI,OperationalChann_A_LI>>
F (Asign_Ab & (ks | ks_LI) & "<<Alice>> F (Asign_Ba| Asign_Ba_LI));

atl "NOR for Bob"
E F (Msg1 & Msg2 & RD_Act_Req & Positive_Result_B & "<<Bob>> F (ks | ks_LI));

atl "NOR for Alice"
E F (Msg1 & RD_Act_Req & Positive_Result_A & "<<Alice>> F (Asign_Ba | Asign_Ba_LI));

atl "Abuse free for Bob"
"<<Dis_Alice>> F(Dis_Alice_Prove2TP & "<<Bob>> F (ks | ks_LI));

atl "Abuse free for Alice"
"<<Dis_Bob>> F (Dis_Bob_Prove2TP & "<<Alice>> F (Asign_Ba | Asign_Ba_LI));
```