SOLVING A GLOBAL SOFTWARE PIRACY PROBLEM

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF PHILOSOPHY

Supervisors

Dr Alan Litchfield

August 2017

By

Jeff Herbert

School of Engineering, Computer and Mathematical Sciences

Copyright

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Signature of candidate

Acknowledgements

I would like to thank my wife Farah, who has encouraged me and patiently supported me through many weekends and late nights away from home completing my research and writings. She has worked as hard as I have to make this research a success: through many life changes, including our wedding, parenting two children and building a house together. I would also like to thank my oldest children, Jordan and Caitlin, who have had less father time than I wanted, and to thank my younger children, Alex and Zara, who have missed me greatly in this last year of research. One day I hope they will be able to read this. And finally, thanks to Alan Litchfield, my supervisor, for all the interesting conversations and helping bring my perspective of industry and academia closer together.

Dedication

I dedicate this study to Farah Herbert, who set my journey into academia in motion by challenging me and convincing me that I would not regret it.

She was, I have discovered, completely correct.

Abstract

This thesis presents a novel method to design a global cross-platform Software License Validation system, using cryptocurrency blockchain technology to ameliorate software piracy and provide controls for software creators to protect their copyright. Protecting software copyright has been an issue since the introduction of desktop computers in the late 1970s and Software License Validation has been a primary method employed in an attempt to minimise software piracy. More recently, the software piracy problem has expanded to include mobile platforms, presenting a distinct challenge for software creators to retain copyright. The objective of this research is to design a system that demonstrates software piracy prevention and provenance of software applications through cryptocurrency blockchain technologies.

Publications

A Novel Method for Decentralised Peer-to-Peer Software License Validation Using Cryptocurrency Blockchain Technology	Herbert, J. & Litchfield, A. (2015). 38th Australasian Computer Science Conference (ACSC 2015) (pp. 27–30). Sydney
Bitcoin & Co: An Ontology for Categorising Cryptocurrencies.	Herbert, J. & Stabauer, M. (2015). M-Sphere: Book of Papers (2015) (pp. 45-55). Dubrovnik

Contents

List of Tables

List of Figures

Chapter 1

Current State of Software Piracy

1.1 Introduction

Since the advent of home computers in the late 1970s, copying software created by others for the popular computing platforms has become a commonplace activity that has evolved with technological advances in computing and connectivity. Software piracy, as it has become known, has resulted in a significant economic loss for the software creators, at an estimated economic cost of US$132 billion per year. This has heralded a new challenge for protecting software copyrighted works, in particular as the Internet and smartphone have evolved.

Software license authorisation has become the primary software piracy prevention and provenance method for software vendors. However, this can be costly and complex, and has limited effectiveness as current software license authorisation methods can be relatively easily overcome or bypassed. These issues reduce incentives for smaller independent software vendors and software creators to protect their software, particularly in respect to the Microsoft Windows and Android platforms.

The objective of this research is to explore the feasibility of using cryptocurrency blockchain technologies to enable software piracy prevention and provenance and

provide the controls for software creators to protect their copyright. To this end, this research proposes a novel method to provide a globally ubiquitous Software License Validation (SLV) method using cryptocurrency blockchain technology over a distributed computing model.

1.2 Evolution of Software Piracy

1.2.1 The 80s – Software Piracy Genesis

Software piracy, abbreviated as "piracy", commonly describes the copying or use of computer software in violation of its license. It has become a long standing technology issue, since the advent of the hobbyist home computer (Maude & Maude, 1984; Mooers, 1977) and the introduction of the personal computer in business (Suhler, Bagherzadeh, Malek & Iscoe, 1986).

Thriving pirate communities were built around the popular home/desktop computers of the 1980s. The methodology for piracy was the duplication of the storage medium (typically cassette tape or floppy disk), and distribution was facilitated through Bulletin Board Systems (BBSes) using modems. This allowed users to digitally exchange software files, and piracy became an international phenomenon (Holsapple, Iyengar, Jin & Rao, 2008). In 1985, the Yellow Book CD-ROM standard was released, providing a significant increase in storage capacity for the medium (ISO/IEC, 1996). In addition to the capacity benefits, software companies used customised CD-ROM media to prevent easy duplication, creating a technology barrier to piracy. This barrier was overcome by the early 1990s with the release of re-writeable CDs (CD-R) (Holsapple et al., 2008) allowing any actor to engage in piracy through cost-effective duplication of CDs for illegal distribution and resale.

1.2.2 The 90s – Rise of the Internet

In parallel with the evolving optical media in the early 1990s, a number of new technologies emerged that would accelerate piracy considerably (Athey & Stern, 2013). The Internet evolved from the ARPANET network into a public network that provided easier file transfer mechanisms such as File Transfer Protocol for distribution of pirated software. The invention of the World Wide Web by Tim Berners-Lee (Zimmermann, 2012) led to an increase of piracy through web sites known as “warez” sites (Chaudhry, Chaudhry, Stumpf & Sudler, 2011). These sites provided an enhanced capability for actors to share pirated software, license files, software cracks and keygens (software to re-generate licenses) (Kammerstetter, Platzer & Wondracek, 2012).

In the late 1990s, the introduction of Digital Subscriber Line and Cable improved digital download speeds by up to 2,600%, which further enhanced piracy by reducing download times and making pirated software rapidly available. Internet bandwidth would continue to grow (Nielsen, 2014), and the rate of piracy would continue to increase (Business Software Alliance (BSA), 2004, 2006) due to the ease of access to illegal software. Warez operators started to experience technical and commercial issues with the growing demand for pirated software, creating capacity and scalability pressures on web servers and Internet connections. The development of "anonymous" peer-to-peer (P2P) file sharing technology, such as Napster, Gnutella, LimeWire and Kazaa, solved these issues by enabling direct piracy between actors, bypassing the requirement for warez websites. P2P file-sharing quickly became a popular vector for piracy, but was short-lived due to successful litigation of the P2P software vendors and actors involved in the file-sharing of illegal software (Goldman, 2004).

1.2.3 The 00s – P2P and Mass Storage

By the mid-2000s, a new P2P file-sharing protocol called BitTorrent had been developed to overcome the issues that resulted in the Napster litigation, re-activating the P2P piracy vector (Chen, Chu & Li, 2014). BitTorrent splits files into segments, encrypts each segment, and provides actors with a distribution mechanism for the segments across the P2P network. This process subsequently protects the user from legal jeopardy, as actors are only sharing encrypted segments. However, BitTorrent trackers, the necessary index service for BitTorrent, have instead become the focus of litigation – although this is considered controversial, as BitTorrent trackers do not hold pirated software themselves (Kigerl, 2013). To further reduce liability, distributed tracker software such as Azureus/Vuze and Mainline became available. These introduced the concept of "trackerless torrents" by using a distributed hash table database to protect the identity and IP address of the actors. Work continues to further enhance BitTorrent anonymity to protect BitTorrent trackers from litigation risks. One concept is a "virtual torrent", which is based on the distributed tracker approach and is used to describe some web resource, whilst another implementation "Anatomic P2P" uses a distributed network of nodes that route traffic to dynamic trackers (Meshkova, Riihijärvi, Petrova & Mähönen, 2008).

In addition, the early 2000s gave rise to the Universal Serial Bus (USB) mass storage market, which introduced a high capacity, mobile storage tool for piracy. USB storage increased in capacity through a number of different forms of media over the decade: (i) USB Flash Drives started with a tiny 8MB capacity and grew to 128GB capacity, providing a resilient and easy-to-conceal form factor to store large volumes of data; (ii) Portable USB 2.5-inch hard drives were released, providing a very useful tool for compact storage of large volumes of data, reaching 1TB by 2010; and (iii) DVD and Blu-ray superseded CDROM and CD-R, increasing the data density of the optical media

disk to 8GB and 25GB, providing very cheap long term storage for large volumes of data. These technologies enabled software pirates to pursue the more classical 1980s methods of software piracy, providing bulk storage of pirated software on a single media disc.

1.2.4 The 10s – Mobility and Identity

The evolution of the original hand-held devices, such as the Apple Newton in the 1990s and the Palm PDA in the 2000s, would herald the innovation of the smartphone. The smartphone is a converged device that combines the functionalities of a PDA, cell phone, GPS and camera. It also presents an application development platform and ecosystem that allows developers to easily write software for the smartphone operating system. Smartphones are largely recognised as being launched with the release of the Apple iPhone in 2007, whilst Google's Android operating system, also released in 2007, grew rapidly through too 2010, when it overtook Symbian to lead the smartphone market. Smartphones would introduce a new platform opportunity for piracy, despite the fact that most smartphone Apps are free or very low cost in comparison to commercial-off-the-shelf (COTS) software. Smartphone piracy rates can reach 90% on the Android platform (Davies, 2013; KeyesLabs, 2013; Rasch & Wenzel, 2015; Smith, 2015): much higher than desktop software (Business Software Alliance, 2014a). In respect to the Apple ecosystem, although Apple has managed to help developers overcome piracy issues through mechanisms in the app store (Claburn, 2009; Goodin, 2014), hackers consistently find a way to overcome new copy protection mechanisms (Haley, 2014; Panzarino, 2013).

Many software vendors now utilise a cloud Software-as-a-Service (SaaS) deployment model to facilitate software piracy prevention and provenance, where users subscribe to SaaS with their credentials to access their software. However, a new form of

piracy has resulted – identity piracy. This form of piracy occurs where user credentials are either shared between users within organisations, or have been obtained through malicious attackers compromising user accounts (Chabinsky, 2015). BSA research has shown that 52% of users say they share credentials (Business Software Alliance, 2014a). The BSA categorises credential sharing as a form of "per seat" under-licensing, and as such is considered piracy. The BSA also notes "as cloud services penetrate emerging markets, expect the incidence of credential sharing to go up".

1.2.5 Impact of Software Piracy

The impact of commercial economic loss from software piracy and its breach of copyright is recognised globally. Analysing the annual BSA piracy reports from 2004 through to 2014 shows that an estimated US$507 billion dollars has been lost to piracy, with worldwide desktop piracy rates growing from 35% to 43%. Although all platforms are experiencing piracy, there is, however, very little peer reviewed research investigating the cost of piracy on non-desktop platforms. Arxan Technologies (2015) have estimated gaming industry losses at US$74 billion in 2015, and assuming gaming piracy follows a similar piracy growth trend to desktop piracy across the 2004 to 2014 decade, losses for the gaming industry would be estimated at US$524 billion. Combining this economic cost with the BSA report results in a cost from software piracy over 10 years at over one trillion U.S. dollars, excluding software piracy from the mobile platforms or cloud services.

1.3 The Blockchain

The blockchain is a distributed ledger technology developed by Nakamoto (2008), and first introduced with the creation of Bitcoin, the first cryptocurrency. Cryptocurrencies are a new form of digital currency that are distributed peer-to-peer electronic cash

systems and are the first technology to successfully overcome the requirement for a centralised party to validate transactions. The cryptocurrency and blockchain architecture provides several blended features that compromise a cryptocurrency ecosystem including: (i) cryptographic validation for all transactions; (ii) decentralised money; (iii) the minting of bitcoins; and (iv) transaction processing functions: all stored on public ledgers within a quasi-anonymous framework (Brikman, 2014).

Blockchains are a new data structure that are cryptographically secured and distributed across a network. The technology supports cryptocurrencies such as Bitcoin, and the transfer of any data or digital asset. Blockchains are economic systems that achieve consensus among distributed nodes, allowing the transfer of digital goods without the need for centralised validation of transactions. The present blockchain ecosystem is, like the early Internet, a permission-free innovation environment in which email, the World Wide Web, Napster, Skype, and Uber were built. The domain, touted as a disruptive technology (Underwood, 2016; Ford, 2014; Noyen, Volland, Wörner & Fleisch, 2014), is expected to mature over time, with industry leading the research and development.

Inside the blockchain is a database that is a public ledger of immutable transactions, cryptographically secured and distributed across a peer-to-peer network running over the Internet with many practical use-cases. The database is a chain of transactions, formed into blocks, that provides non-repudiation of previous transactions, and hence, transactions can never be changed without breaking the chain of blocks. The cryptocurrency ecosystem is a network of peer-to-peer participants called Miners, all of which have the blockchain public ledger. Together, they provide a consensus approach for validating transactions, thereby eliminating the requirement for a central party. Cryptocurrencies overcome the necessity of a centralised "trusted authority" (Nakamoto, 2008), and thus remove or significantly reduce transaction fees associated with transactions, such as those incurred with commercial banking transactions.

There are many different use-cases for cryptocurrency and blockchain technologies.

The distributed nature of the blockchain, combined with the ability to verify transactions, has led to the development of second generation blockchain protocols that have the capability to perform new functions, supporting innovative on-blockchain services such as storing data and new scripting capabilities (Bradbury, 2014; Buterin, 2017). Cryptocurrencies also provide an inherent anonymity for their transactions, and data stored on the blockchain is mostly encrypted to provide confidentiality in the public ledger environment. Hence, cryptocurrency and blockchain technologies can be utilised in different forms by software creators.

Herbert and Stabauer (2015) propose a cryptocurrency model categorising cryptocurrencies, and the different implementations of the cryptocurrency blockchain, for use by applications. This model considers the characteristics of cryptocurrencies and presents examples of the use-case applied to a range of cryptocurrencies. Native Blockchain Applications (NBAs) provide additional application functionality built into the cryptocurrency or blockchain. This capability allows the software engineer to customise the cryptocurrency or blockchain to meet a set of specific requirements to achieve the outcome desired. However, NBAs require careful consideration as there are many non-functional requirements that must be achieved through the software engineering process.

1.4 Software License Validation

Historically, many different methods have been proposed or implemented in an effort to prevent software piracy and to protect software creator copyright. There are many circumstances in which software piracy may occur, and many forms of software piracy over many technology platforms. There is currently a gap in protecting software creator copyrighted works, and there is no mechanism that allows the software creator to enforce their copyright globally without requiring costly legal enforcement.

The review of literature on software piracy reveals that the user is the endpoint for all piracy, and that focussing on user identity is the key to software piracy prevention. It concludes that binding user identity to software entitlements is essential to authorising access to software and for providing an important provenance function. However, there is no disincentive to sharing credentials that provide access to software, and the software itself is not protected from other forms of piracy once the initial user authentication and access to software has been completed. Software-at-rest is particularly vulnerable to threat actors, who will attack the code structure directly to defeat prevention or protection mechanisms.

This research contends that SLV using blockchain technologies is feasible for achieving the outcome of cost-effective software piracy prevention and provenance. The proposed SLV method, called ReSOLV, uses blockchain technology to address the main problems identified in literature as follows:

- It needs to be global
- It needs to support multiple platforms
- It needs to be cost-effective
- It places license authorisation in the hands of the software creator
- It protects software-at-rest
- It uniquely binds user identity to software entitlements
- It disincentivises credential sharing
- It provides end user anonymity and data confidentiality

To support these claims, this research presents artefacts from the Literature Review, Methodology, Requirements Engineering (RE), and Functional Decomposition (FD) chapters as described in the following section.

1.5 Thesis Structure

Using a Design Science Research (DSR) methodology, this research aims to utilise blockchain technology to provide SLV, where software entitlements are delivered through data stored cryptographically as transactions in the blockchain. This will require defining requirements and creating use-case specific data flows for the purpose of SLV.

The outcome of the research is to design a system that demonstrates that it is feasible to use a Native Blockchain Application (NBA) for the purpose of SLV. This thesis is structured as follows:

Chapter 1: Introduction. This chapter is the introduction to the subject of software piracy, SLV, and the cryptocurrency blockchain.

Chapter 2: Literature Review. This chapter presents a literature review that explores two separate subject matters, which will be brought together in this thesis: software piracy, and the cryptocurrency blockchain.

- **Current State of Software Piracy**

  Section 2.2 through 2.4 review the current state of software piracy and explores the scope of software piracy issues across platforms. There are distinct questions that have emerged; what is software piracy, who exactly are the software pirates, and how do they relate to the different methods used to protect software? The methods of software piracy prevention are discussed, and a high level software piracy model is presented, which provides an insight into "who" is undertaking the breach of copyright, "what" the breach of copyright is, and "how" the breach occurs. The Software Platform Vulnerability Lifecycle is introduced, and related issues of enforcement and the relationship between software piracy and malware are also examined.

- **Blockchain Technology**

  Section 2.5 presents the concept of the cryptocurrency blockchain and describes the characteristics that demonstrate its benefits for a globally ubiquitous distributed application, such as SLV. Prior research and related proof-of-concepts are discussed and explored in the context of SLV. This chapter concludes with hypotheses and research questions.

Chapter 3: Methodology. This chapter outlines the DSR approach that this research will use. It identifies the intersections that cryptocurrency and blockchain technologies have with existing knowledge domains, and presents a brief overview of the expected requirements for SLV in order to better determine the appropriate Research Methodology. A DSR framework is introduced, outlining research contribution types. This establishes that the blockchain-based SLV research is classified as an invention, and is considered a valid DSR research opportunity. Software engineering methodologies, including Agile, Behaviour Driven Development, and Agent Oriented Systems Engineering, are introduced and briefly discussed.

Chapter 4: Requirements Engineering. This chapter discusses the approach to RE for the ReSOLV method. Definitions and the general RE process and activities are established. The ReSOLV model is presented, demonstrating the main constructs for SLV on the blockchain. The requirements specification establishes functional and non-functional requirements for the ReSOLV model, culminating in the design of the ReSOLV High Level Architecture (HLA) and Reference Architecture (RA), which describe the core functionality of the ReSOLV method.

Chapter 5: Functional Decomposition. This chapter presents a FD of the ReSOLV RA, to identify what processes need to be built. Data Flow Diagrams (DFDs) and pseudocode are introduced for two of the main systems established in the

ReSOLV RA. These artefacts explain how the functions and processes transform data, and describe the data flows that demonstrate that the SLV method is feasible.

Chapter 6: Discussion. This chapter recaps the motivations for this research and reviews the research findings for SLV and discusses the relevance of the artefacts as outcomes of the research. Each Research Question is addressed with researched evidence artefacts supporting confirmation of the research questions. RQ1 concludes that NBAs are suitable for the ReSOLV SLV method, whilst RQ2 confirms that user anonymity and data confidentiality are achieved using blockchain-based technologies for SLV. Reflections on DSR are provided, discussing how effective the methodology was for this type of iterative development research, and commenting on improvements for future research efforts.

Chapter 7: Conclusion. The conclusion summarises the objectives for this research and the findings, concluding that the researcher is comfortable with the hypotheses. It also explores limitations and future research opportunities.

Chapter 2

Literature Review

2.1 Introduction

This chapter presents a review of literature of two domains: software piracy and blockchain technology.

Sections 2.2 through 2.4 examine the scope of software piracy to identify the problems, actors and processes that surround software piracy. They seek to establish the common difficulties in protecting software from piracy, and ultimately identify the underlying "lowest common denominators" for a cross platform method to prevent software piracy.

Section 2.5 provides an overview of the functionality and characteristics of cryptocurrency and blockchain technologies, whilst Section 2.6 explains a cryptocurrency taxonomy and demonstrates how blockchain technologies can be applied to distributed systems. It seeks to establish the capability of the blockchain technology to provide the required "lowest common denominators" from the software piracy review of the earlier sections.

The chapter concludes, in Section 2.8 on page 90, with discussion of the feasibility of utilising blockchain technologies to prevent software piracy, and outlines the Hypotheses

and Research Questions for this research.

2.2 Platform Specific Piracy

2.2.1 Desktop Platforms

Desktop computing began in the late 1970s and early 1980s with several well-known brands establishing their products, including Acorn, Apple, Atari, BBC, Commodore, IBM, Sinclair, and Tandy (Computer History Museum, 2016). Over the next three decades, Apple, and the widely cloned IBM PC, would be the only survivors in the highly competitive hardware manufacturing environment, with the operating system becoming the dominant differentiator for each of the remaining hardware platforms.

As of December 2015, analysis of the current market share of the operating systems shows that Microsoft has by far the largest number of installations and number of torrents, as shown in Figure 2.1 (Netmarketshare, 2015; Steam, 2015; Van der Sar, 2009). Hence, based on quantity relative to operating systems and software from other vendors, it is expected to have the largest piracy issue.

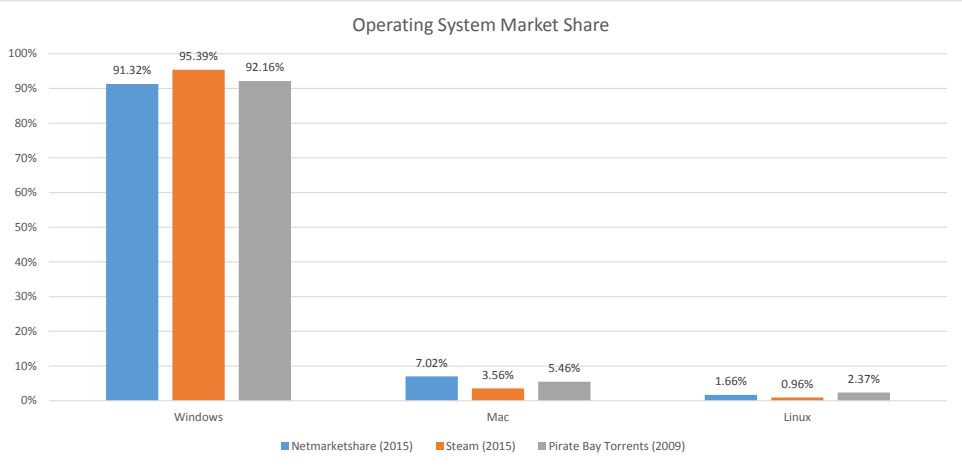


Figure 2.1: Desktop Operating System Market Share (2015)

There is limited objective research available that quantifies the rates of desktop piracy, especially in the context of quantification of piracy by platform. The majority of research refers to the BSA reports, acknowledging a potential industry bias in quantifying the scope and losses that result from piracy. Furthermore, the BSA reports only pertain to the category of commercial desktop software [1,2],and excludes other categories such as gaming software. The most recent report from the BSA estimates that, in 2013, 43% of software on desktop computers around the world was not properly licensed, with a commercial value of US$62.7 billion (Business Software Alliance, 2014a).

Gaming software industry trade organisations claim piracy rates of 90% (Senior, 2012). However, investigation of piracy rates in the industry shows that most methods for analysing piracy use BitTorrent activity for piracy measures (Solon, 2013; Van der Sar, 2009). Unfortunately, using BitTorrent as the method for quantifying piracy creates inaccuracies due to over-accounting of downloads, resulting in an over-stated piracy rate (Solon, 2013; Ployhar, 2012). Similarly, Drachen, Bauer and Veitch (2011) find that quantifying game piracy as a dollar value representing loss of sales is difficult due to the challenge of reliably identifying unique BitTorrent peers. Hence, a lack of accurate data means that quantifying piracy rates and estimating commercial value loss in the gaming industry is difficult.

Van der Sar (2009) uses BitTorrent statistics from pirate site "The Pirate Bay" to provide an estimation of the proportion of piracy between Windows, MAC and Linux operating systems. The analysis shows that out of a total of 53,541 torrents, the Windows torrents have a 92.16% (49,345) share, as compared to the Mac 5.46% (2,925) and Linux 2.37% (1,271). Despite the difference in time frame between the desktop operating system market share in 2015 and the BitTorrent analysis undertaken

[1] Desktop: includes desktop or laptop computers running Windows or Mac operating systems
[2] Software: includes operating systems and Commercial off-the-shelf (COTS)

in 2009, it is shown that piracy rates correlate to operating system market share, and that Microsoft Windows clearly has the most significant piracy problem due to its market share.

From a geographic perspective, although piracy rates in first world countries is decreasing, the BSA reports that developing countries show much higher piracy rates (Business Software Alliance, 2014a). Developing countries maintain the highest piracy rates because the end users cannot afford to pay for the software (Asongu, 2014), or, despite being signatories to World Intellectual Property Organisation (WIPO) (Lu & Weber, 2009), the cultures of some countries accept piracy as the norm.

Contrary to this, however, Athey and Stern (2013) analyse Windows Automatic Update telemetry data and conclude that, rather than income of the individual, the quality of the institutional and infrastructure environment of a country correlates with the rate of piracy. For example, countries with better broadband infrastructure show higher piracy rates (Athey & Stern, 2013), and if this is the case, the proposed piracy prevention strategy of regional pricing (Asongu, 2014) may not have any effect on the rate of piracy (Athey & Stern, 2013).

There are a number of mechanisms used by desktop software vendors to prevent piracy. The primary copy protection mechanism is a license key that is used during the software installation process to provide the end user entitlement to install the software. Another mechanism is Digital Distribution Systems (DDS), where software is downloaded online. There are several disadvantages to relying on these mechanisms to protect software. Software pirates simply need to copy the digitally distributed media and publish it with the appropriate license key to negate any benefit DDS has offered. Furthermore, for more advanced license key protection mechanisms, software crackers can create key generators, or "keygens", to generate licenses that the software installation process will accept (Sigi, 2010), again easily overcoming the copy protection and making DDS irrelevant. Alternatively, software vendors may implement

an activation key mechanism, where the customer is required to authorise the software via online activation once the software is installed. However, depending on the type of key, these activation keys may be used many times. As with license keys, a software pirate just needs to pair the digitally distributed media and publish it with the appropriate license key. Although activation keys will help reduce illegal installations of software, the protection mechanism is easily overcome by obtaining a new key. Neither method offers any further protection mechanism, once the digital media and key are obtained, and the software installed. In addition, software crackers have developed techniques to masquerade authorisation services through custom configuration of Domain Name Service (DNS) records, defeating the authentication mechanism.

2.2.2 Mobile Platforms

With the advent of the iPhone and Android platforms in 2007, a new vector for piracy was created: mobile applications, or "Apps". Apps have grown significantly, with Statista (2015) showing 1.5 million Apps listed on the Apple App Store, and 1.6 million on Google Play, as at July 2015. Netmarketshare (2015) reports that the Android ecosystem has the largest share at 57.29%, as shown in Figure 2.2, whilst the Apple iOS ecosystem comes in at second, contributing to 35.43%.

Despite the low cost of Apps, piracy is a considerable issue for mobile platform developers. In the Apple ecosystem, Khan, Othman, Ali and Madani (2014) state that 84 of the top 100 iOS applications are affected by piracy, primarily through jailbroken devices. For Android Apps, the rate of piracy is particularly high, with Android developers experiencing a consistent 90%-99% piracy rate over the last two years – the games category having the highest piracy rates (Davies, 2013; KeyesLabs, 2013; Rasch & Wenzel, 2015; Smith, 2015).

Although Digital Distribution Services (Han & Shon, 2014) are intended as a method

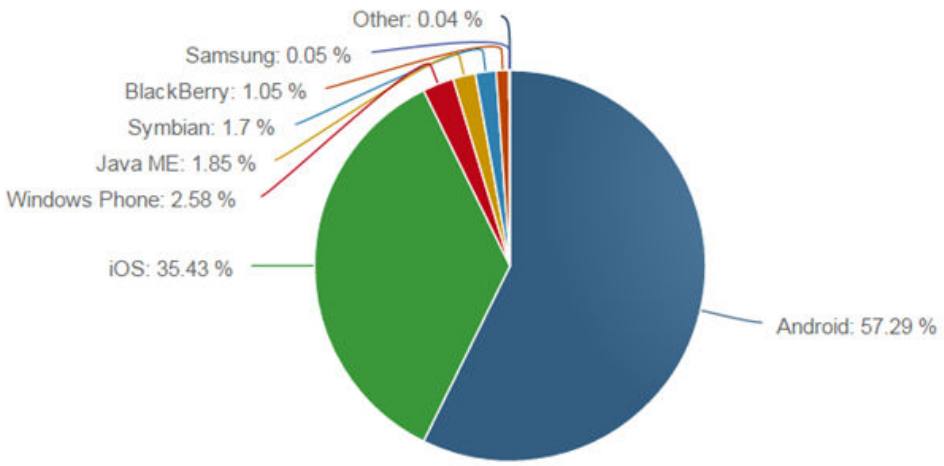


Figure 2.2: Mobile/Tablet Operating System Market Share (Netmarketshare, 2015)

to reduce piracy opportunities, the mobile App stores still have challenges in respect to piracy. Apple still faces consistent challenges, with crackers discovering new exploits in the Apple Store (Goodin, 2014), and copying iOS Apps on jailbroken iPhones (Panzarino, 2013). On the other hand, due to the ease of copying and distribution of Android Apps, the Google Play DDS for Android, and the open Android platform itself, exacerbate piracy in the Android ecosystem (Ravenscraft, 2012; KeyesLabs, 2013; Gurulian, Markantonakis, Cavalaro & Mayes, 2016).

The piracy rate on mobile devices is significantly higher than on desktop computers, despite the fact that mobile device Apps are usually cheaper than COTS desktop software. With only 16% of Android users willing to pay for Apps, Choi, Au and Liu (2014) find that the mobile App market is characterised by a low willingness to pay, despite the relatively low cost, supporting the reported high piracy rate on the Android platform. There is anecdotal research on the geographic distribution of App piracy. KeyesLabs (2010) provides an analysis of Android piracy by country for a single App, and shows mobile piracy rates over 66% for most first world countries.

Lastly, in addition to the problem of piracy, new issues have been identified that

have resulted from pirated mobile Apps. Pirated mobile Apps now present a new attack vector for criminals, through hackers pre-infecting an App with malicious code and re-releasing the malware infected App (Computer Fraud and Security, 2011) for unsuspecting users to download (Yulong Zhang, Zhaofeng Chen, 2015).

2.2.3 Software-as-a-Service

Business Software Alliance (2014a) highlights the impact of cloud computing on licensed and unlicensed software use. Public cloud SaaS delivery of software functionality, online over the Internet, is intended to reduce unlicensed software use through a subscription-based model. It requires users to authenticate to a web portal to run the software solution. However, the analysis by the BSA shows that SaaS model, encourages credential sharing. Despite the fact that SaaS models show lower unlicensed software use, the BSA indicates that 52% of online credentials are being shared. Therefore, the BSA concludes that SaaS models are not expected to have a significant impact on rates of piracy. In addition, some SaaS vendors allow customers to install the software on rental virtual infrastructure, a model known as Infrastructure-as-a-Service (IaaS). In this model, software is more vulnerable to piracy due to the ease of deploying software across multiple virtual instances (Wang, Cheng, Cho & Wang, 2015). Hence, a new license model will be required to overcome this challenge.

2.2.4 Video Game Consoles

As video game piracy grew with the consumer adoption of computer games at home, console platforms from Sega (1984), Nintendo (1984), and Sony PlayStation (1994) were introduced and had a significant effect on preventing game piracy (Depoorter, 2014; Ployhar, 2012). This was achieved because the consoles used game cartridges with chips to store the game software. The cartridges were too expensive to duplicate, so

game pirates focused instead on modifying ("modding") the game consoles themselves to accept unlicensed games (Tsotsorin, 2013). However, the associated costs of the cartridge manufacturing, the limited capacity of the cartridges, and availability of cheap storage alternatives, such as CDROM optical media, resulted in the console manufacturers moving to optical media for game storage. For example, Sony PlayStation and Microsoft Xbox have implemented custom encryption and obfuscation software and hardware techniques to prevent piracy (Khandelwal, 2015; Lejacq, 2013). However, both Microsoft Xbox and Sony PlayStation consoles have been successfully hardware modded to enable piracy, leading to definitive responses from both companies. Microsoft banned one million users with modded consoles (Ionescu, 2010), whilst in 2011 Sony shut down their PlayStation Network as a result of a jailbroken PlayStation3 hack, which not only allowed pirated games but led to one of the largest data breaches in 2011 (Lynley, 2011). Hence it is shown that gaming consoles can be "jailbroken" to achieve piracy, although the risk of detection and the consequences deter mainstream consumers from piracy (Sony, 2016a, 2016b).

The most significant game piracy threat to the video game console game publishers is that of the used games and the second hand market, which operate in breach of the game End User License Agreement (EULA). Most game publishers only grant a license to use the game software, and expressly prohibit rental or resale (Electronic Arts, 2016; Sony, 2016b; Ubisoft, 2016).

2.2.5 Other Platforms

Anywhere software is developed, the opportunity for piracy exists. In addition to discrete software, the scope of piracy targets extends to many different, and largely unexplored, areas where piracy may occur but has little data or research. To summarise other forms of piracy, the following further types of platforms have been identified as

potential targets, where piracy may have occurred, or which are vulnerable to piracy in the future as technology changes and develops.

1. Read-only memory chip (ROM) piracy was occurring from the mid-1980s and the opportunity for piracy in the context of breach of copyright exists today (Newman, 2013; Folsom, 1985). Early examples of ROM piracy include the IBM PC BIOS copyright breaches, where IBM clone computer importers, copying the IBM PC basic input/output system (BIOS) into clone PC's, were successfully sued for breaching IBM copyright (Moon, 2009).
2. Software libraries that software distributions use are likely targets for piracy. Many businesses create and sell libraries, which are then used with common development platforms. These can be easily pirated and used in breach of the license agreement.
3. Emulators are third party software that mimic the function of a hardware device: often legacy proprietary hardware such as Radio Shack TRS-80, Commodore 64, or Nintendo Gameboy. They provide the opportunity for users to use and enjoy software developed for these systems despite the hardware not having been manufactured for decades. Although emulators are legal within the United States, unauthorised distribution of copyrighted software used by the systems (examples include games, BIOS or firmware code), remains illegal (Karas, 2001; Newman, 2013).
4. Applications that execute in a virtual machine environment locally to the user, such as the Java Virtual Machine (JVM), or other markup languages such as HTML5, may be vulnerable to piracy, as the code executed on the local machine may be copied.
5. Scripting languages with visible source code, such as AppleScript, JavaScript, Python, Ruby, PHP, and macro languages, such as Visual Basic for Applications,

are vulnerable to piracy due to the direct execution nature of the interpreter (Patel & Pattewar, 2014; Sharma, Sharma & Tyagi, 2015). Similarly, job control languages and shell scripts are easily copied.

6. Applications that run on web servers and within application frameworks, such as JVM, Ruby on Rails, and .NET, may also be vulnerable to piracy (Sharma et al., 2015).
7. Virtual Machine or Cloud based applications are vulnerable to piracy, as the underlying virtual machine disk file, containing the operating system and software, can easily be copied and utilised (Baumann, Peinado & Hunt, 2014; Wang et al., 2015; Singh, Jeong & Park, 2016).
8. Containers is a concept where applications can be run independently of any host operating system and it's libraries (Kang, Le & Tao, 2016; Jaramillo, Nguyen & Smart, 2016). The objective of Containers is application portability, independent of the underlying operating system, and it is purpose-built to allow software to be easily replicated and executed on another Container host, thereby creating a potential piracy threat.
9. Internet of Things: The number of embedded platforms and Internet of Things devices is expected to grow rapidly as technology in the mobile, wirelessly connected world continues to evolve. Examples include devices such as smart cities, car computers, and wearable technology on every human being. Due to the distributed and easily accessible nature of these devices, any embedded platform may be vulnerable to piracy (Kumar, 2017; Singh et al., 2016).

2.2.6 Effect of Digital Distribution Services

This section discusses DDS, and whether DDS has had any effect on reducing piracy. DDS is now commonly used by vendors, in preference to physical media. It requires

end user authentication to prove identity, prior to downloading software, and may be successful in reducing piracy by reducing the opportunity to copy physical media and keys.

DDSs are used to provide an easy mechanism for legitimate purchase of software and to reduce piracy. As discussed earlier, the mobile and video game platforms natively provide DDS. Google and Apple have their own DDSs for mobile Apps, through portals such as Google Play and the Apple App Store. Companies such as Microsoft and Sony offer DDS for console games, respectively through Xbox Live and PlayStation Network. Independent DDS, such as Steam, Amazon, Gamersgate, and GoG, provide licensing and Digital Rights Management (DRM) to a gaming specific demographic. For the most part, DDS is focused on secure software delivery (via download) and record-keeping of the license entitlements of the customer, rather than prevention of piracy.

DDS for Desktop can be grouped into three categories: (a) enterprise software vendors (ESV); (b) independent software vendors (ISV); and (c) games software vendors (GSV). Whilst ESVs such as Microsoft and Adobe have developed their own DDS through portals [3], most of the ISVs do not utilise DDS mechanisms for distribution or to prevent piracy. This lack of utilisation of DDS for Desktop may result from a commercial cost-benefit decision by the ISV. The complexity of protecting COTS software piracy from the range of piracy threats means that is likely to be a high cost investment. An ISV needs to implement several technical piracy prevention mechanisms, such as: (a) installation license keys; (b) activation keys that require authentication; and (c) DRM for customers. The ISV also needs infrastructure and management to provide the DDS. It seems that ISVs either do not perceive a value in DDS, or do not have a suitable business model to utilise a DDS.

However GSVs are highly invested in DDS. This section specifically examines the Steam online DDS strategy, and its impact on piracy prevention. Steam is the market

[3] Microsoft Office365 portal; Microsoft “Volume License Server Centre”; and Adobe “Creative Cloud”

leader in the games software digital distribution marketplace, with an estimated 50%-70% market share (Chiang, 2011). Steam provides services to both software publishers and the players themselves. The Steam approach and strategy to reduce game piracy is to offer value added services to the player base, such as peer-to-peer gaming and social networks. Players are incentivised to purchase games through Steam, despite the fact that the software is subject to DRM, because pirated software cannot match the value of the services that Steam provides. Tsotsorin (2013) discusses Steam's statement that they have been very successful in curbing game piracy in Russia. The fundamental lesson that Steam provides, is that game piracy is a service issue. The Russian market, with one of the highest rates of piracy in the world, is now Steam's largest continental European market.

Steam protects against piracy via the Steam Anti-Piracy suite, which includes Custom Executable Generation (CEG), DRM, and Valuable Platform-Dependent Features (Steam, 2016). Use of a Steam client is also required to run on the player's computer. CEG is a mechanism that creates a unique build of game software for each player when the player purchases the software. This is then linked to the player's Steam account, and the player must authenticate to Steam through the Steam client to run the game software. DRM protects the game software through licensing and encryption, providing secure distribution mechanisms for the game publisher, including encryption, whilst the software is in transit, and encryption of the software itself using a private key belonging to the player. On the player's computer, the Steam client decrypts the software during installation (Bilger, 2006), protecting the install package from being copied. CEG-enabled games, providing the highest level of protection against piracy, are executed from the Steam agent installed on the computer, and require Internet access to Steam to login and play the game, making it much harder to pirate the downloaded software. Steam is cross-platform, providing services to Windows, Mac and Linux clients.

However, Steam games are still potentially vulnerable to reverse engineering and attacks aimed at defeating the Steam DRM, with plentiful discussion on various forums and Github projects (Atom0s, 2016; SteamRE, 2016). Despite Steam's claims of success at reducing piracy, Ployhar (2012) points out that nearly 30% of Steam players have a BitTorrent client installed (according to the Steam Hardware & Software survey taken in 2012: Steam now do not collect data on installed software in their survey), indicating that there is still a likelihood that Steam players continue to engage in software downloads from other sources.

Overall, it is shown that the value added social aspect for gaming encourages legitimate purchase and discourages piracy, and that the gaming network that also provides value added entertainment services is more successful at reducing piracy. Steam's DDS and anti-piracy features provide considerably improved measures to prevent game piracy, but is only available for the gaming demographic.

2.3 Taxonomy of Software Piracy Types

This section explores several aspects of software piracy, with the purpose of identifying the types of software piracy and the piracy actors involved. Firstly, a definition of software piracy is established through industry literature, treaties, and statute law. The types of software piracy are defined, and the actors who undertake the various forms of piracy are identified. Associations between types and actors and roles of piracy are discovered, and from this a taxonomy of software piracy types is established.

2.3.1 Definition of Software Piracy

Intellectual Property covers three distinct sets of rights: copyrights, patents and trademarks (Besen & Raskind, 1991). In respect to intellectual property protection, piracy, the unauthorised use of copyrighted goods, is of serious concern for copyright holders

(Andrés, 2006). There are many public definitions for piracy, which articulate piracy to different levels of exactness, including UNESCO (2015), Techopedia (2015) and Business Software Alliance (2014b).

1. **UNESCO** defines piracy as: "the reproduction and distribution of copies of copyright-protected material, or the communication to the public and making available of such material on on-line communication networks, without the authorisation of the right owner(s) where such authorisation is required by law" (UNESCO, 2015).
2. **Techopedia** is an online expert resource that aims "to help you understand technology" and claims to have one of the web's most comprehensive computer dictionaries. Techopedia has been used in this instance to provide a contrast to definitions from the larger more formal sources. Technopedia's definition is: "Software piracy is the stealing of legally protected software. Under copyright law, piracy occurs when copyright protected software is copied, distributed, modified or sold. Software piracy is considered direct copyright infringement when it denies copyright holders due compensation for use of their creative works." (Techopedia, 2015).
3. **Business Software Alliance** has a comprehensive definition of piracy, but summarises the term as: "the unauthorised copying or distribution of copyright software, including downloading, sharing, selling, or installing multiple copies of licensed software". It defines five primary types of piracy, with sub-categories to articulate specific methods within the primary type (Business Software Alliance, 2014b).

UNESCO (2015) also makes the point that "Piracy" is the popular term used to describe the phenomenon, and states "national copyright legislations generally do not include a legal definition". The website states, "Today, the only international

legal instrument in the copyright arena which provides a definition of "piracy" is the Agreement on Trade-Related Aspects of Intellectual Property Rights from the World Trade Organization" (World Trade Organization, 1994 (Art.51, n.14)). However, further guidance can be provided in jurisdictions such as the European Union. Directive 2009/24/EC enacts specific aspects of rightholder protection into law that correlate directly with the UNESCO definition by criminalising specific acts of piracy, and provides remedies for action[4].

This section has discussed organisational definitions of software piracy. However it has excluded statute law and international agreements. The following section explores how software piracy is defined in statute law, treaties, and agreements.

2.3.2 Treaties, Agreements and Acts

Several international treaties and agreements, as well as Acts within established sovereign domains[5], form the legal basis for defining the actions that result in a breach of copyright or intellectual property rights.

These legal instruments address separate forms of breach of copyright and intellectual property, but fall short of providing a consistent definition of software piracy, or identifying actors that engage in software piracy. The focus is primarily on actions or methods that constitute a copyright breach. This section compares and contrasts the Trade Related Aspects of Intellectual Property Rights (TRIPS) Agreement, and the WIPO Copyright Treaty, the two primary instruments for protecting copyright and intellectual property. It also provides some perspective on how sovereign states have enacted these agreement provisions into law. This investigation is relevant to the examination of software piracy, because the goal of the legal instruments is to protect the rights of the rightholder. However, success depends on nations being signatories to

[4] The European Parliament and the Council of the European Union (2009)
[5] Examples include the US Digital Millennium Copyright Act, and the NZ Copyright Act (1994)

the legal instrument, enacting the protection clauses into law, and having the motivation to police software piracy and enforce copyright. Furthermore, inconsistencies between the instruments, fast moving technology, and new piracy methods can lead to legal protections rapidly becoming obsolete.

The TRIPS Agreement is Annex 1C of the Marrakesh Agreement Establishing the World Trade Organisation, and is a comprehensive multilateral agreement on intellectual property (Nill & Shultz, 2009). It includes areas such as copyright. Computer programs (Article 10.1), databases, and compilation of data (Article 10.2) are included in copyright, and it also provides that authors of computer programs have the right to authorise or prohibit commercial rental or copies (Article 11). However, as the TRIPS Agreement preceded the popular use of the Internet, it does not include mechanisms that enable piracy, focussing mainly on the rights holder as the key to authorising the use of software. Although a new round of TRIPS negotiations was launched at the Doha Ministerial Conference in November 2001, these primarily cover enacting laws, technology transfer, and enforcement of intellectual property rights .

Increased Internet popularity has led to an increase of copying and sharing of digital content, including music, video, and software, mostly in violation of copyright law that was not explicitly covered by the TRIPS Agreement. The WIPO Copyright Treaty is an international treaty on copyright law, concluded in 1996, and provides separate additional protections to the TRIPS Agreement considered necessary due to advances in information technology. Specifically, the WIPO Copyright Treaty ensures computer programs and databases are protected, and outlines rights of rental, rights of distribution, and the right of communication to the public (Sheinblatt, 2014). The WIPO Copyright Treaty was implemented in the United States via the Digital Millennium Copyright Act in 1998 (DCMA), and by the European Union through Directive 2001/29/EC of the European Parliament and of the Council in 2001.

The DMCA increased the punishment for violating copyright using the Internet as

a medium, and also made the creation of software or devices that circumvent access control illegal (Radcliffe, 1999). Owners of software are entitled to reverse engineer software (crack copy protection) under the "fair use" privilege. However it is illegal to create and sell software and devices that undertake this. The DCMA is divided into five titles of which Titles I and II are relevant to copyright protection of computer software:

1. Title I implements the WIPO Copyright Treaty, and enacts copyright provisions into US law, as well as extending the scope of copyright protection to include use of the Internet to access copyrighted materials, and prohibits circumventing protection measures.
2. Title II protects online service providers who may store, index, cache, or have involvement in, transitory communication of copyrighted materials, and where the service provider does not have knowledge of the copyright infringement.

However, there is criticism of the DCMA for being too rigid and taking legitimate rights away from consumers, especially around circumvention of technology controls, such as jailbreaking an iPhone (Swanson, 2010), and rights to free speech (Kretschmer, 2003).

Also in response to the WIPO Copyright Treaty, the EU implemented Directive 2001/29/EC in 2001, enacting similar provisions to the DCMA (Parliament & Council, 2001), but still requiring member states to implement the provisions into law. The Directive also excludes liability for online service providers, who are therefore not liable for the data they transmit, even if it infringes copyright. Unlike the DCMA, the Directive also prohibits circumvention of copy protection measures, making it more restrictive than the DCMA.

New Zealand is party to various international copyright agreements, including:

1. TRIPS Agreement (Annex 1C to the Agreement Establishing the World Trade Organisation 1994);

2. The Berne Convention for the Protection of Literary and Artistic Works 1928 (Rome Act revision);
3. The Universal Copyright Convention 1952.

In respect of New Zealand legislation, rights for copyright protection are provided for in the Copyright Act 1994 and the Copyright (New Technologies) Amendment Act 2008. The Copyright Act 1994 is accompanied by the Copyright Regulations 1995. These Acts define infringement of copyright, acts permitted in relation to copyright works, and include adaptations, which extend liability to providers or transporters of copyright material.

The Copyright (New Technologies) Amendment Act 2008 sought to clarify the application of the Act to digital technology, and to state the provisions of the Act in technology-neutral language. By adding the technology-neutral category of 'communication works', protection of digital technology was extended beyond broadcasts and cable programmes. In 2011 the Copyright Act was strengthened by several anti-counterfeiting provisions, including: (i) empowering border enforcement and criminal authorities to detain and investigate suspected counterfeit goods; (ii) enacting criminal procedures and penalties when wilful piracy or counterfeiting is carried out for commercial advantage; (iii) enhanced ability to obtain authorisation for destruction of fake goods and seizure of the equipment and materials used in their manufacture; and (iv) seizure of the criminal proceeds from piracy and counterfeiting offences.

2.3.3 Types of Software Piracy

Types of piracy can be established and categorised from these various treaties, agreements and sovereign acts. The following list provides an outline of the types of piracy. Items 1 to 5 are defined by the BSA, summarising the objectives of the treaties and agreements in the context of piracy. Item 6 is established as a result of acts that provide

for reverse engineering of software yet also fit into the expanded piracy definitions as outlined by the BSA.

1. End User Piracy

   This occurs when a user reproduces copies of software without authorisation, and includes:

   (a) using one licensed copy to install a program on multiple computers;
   (b) copying disks and keys for installation and distribution;
   (c) taking advantage of upgrade offers without having a legal copy of the version to be upgraded;
   (d) acquiring academic, charity, or other restricted or non-retail software, without a license for commercial use.

2. Client-Server Overuse

   Client-server over-use occurs when too many users on a network are using a central copy of a program at the same time, also known as under-licensing. This falls into two further sub-categories of common license models:

   (a) per device, based on number of devices that have the software installed;
   (b) per user, based on the number of concurrent users running the software;
   (c) credential sharing of cloud-based services, classed as piracy according the Compliance Gap report conducted by the Business Software Alliance (2014a).

3. Internet Piracy

   Internet piracy occurs when software is downloaded from the Internet, regardless of method of purchase. Internet piracy includes:

   (a) Pirate websites, known as warez sites, that make software available for free download or in exchange for uploaded programs;

(b) Internet auction sites that offer counterfeit, out-of-channel, copyright-infringing software;

(c) Peer-to-Peer networks that enable unauthorised transfer of copyrighted programs.

4. Hard-disk Loading

   Hard-disk loading occurs when a business that sells new computers loads illegal copies of software onto hard disks. This may also occur through cloning disk images: a common practice in the reseller channel.

5. Software Counterfeiting

   This is the illegal duplication and sale of copyrighted material, with the intent of directly imitating the copyrighted product.

6. Reverse Engineering

   Reverse engineering, as applied to computer software, normally refers to a variety of practices undertaken to understand how a software program is built and how it achieves its functionality (Lande & Sobin, 1996). Reverse engineering may be conducted legally for a number of reasons that have been established in U.S. legal cases, mainly surrounding fair-use entitlements provided by U.S. Copyright Law. However, it has been established that the legality of reverse engineering is at the discretion of the rights holders, who are increasingly specifying the exclusion of reverse engineering in the software license terms (Electronic Frontier Foundation, 2015). In general, the software license, which must be accepted by the user to install the software, rescinds the rights of the user to reverse engineer the software. The user opts-out of their legal rights by agreeing to the license terms. Hence, users that reverse engineer for “fair use” purposes, or users that reverse engineer software to remove copy protection or license protection (software crackers), may be in breach of the software license terms as defined by the rights holder. This can

be considered a license breach similar to End User Piracy, which is a copyright violation.

2.3.4 Roles of Software Piracy

The types of software piracy acts, as defined by various organisations and legal instruments, have been categorised by the BSA into primary types and sub-types of software piracy (Business Software Alliance, 2014a). Piracy actors can undertake multiple types and sub-types of piracy. However to present a cohesive model of how piracy actors engage in different forms of piracy, this section determines the piracy roles derived from the piracy types, rather than the actors engaging in the types of piracy itself. The purpose of this is to disassociate the actor, engaged in software piracy, from the role(s) that the actor may be actually undertaking during the software piracy process as actors may have multiple roles (discussed in Section 2.4 on page 52).

Each of the types of piracy has a role that undertakes the activities described by the types of piracy. Analysing the types of piracy reveals five primary roles; i) End User; ii) Software Engineer; iii) Cracker; (iv) Distributor; and (v) Profiteer. Figure 2.3 on page 48 visualises the association between types of software piracy and established the piracy roles. Definitions for these roles are as follows:

End User: the End User is a user of the software who breaches the EULA through not-for-profit duplication activities, where a license is misused either deliberately or accidentally.

Software Engineer: the Dictionary of Information Science and Technology (Mehdi, 2013, p.828) defines a software engineer as “professional software developer whose focus is on the software (rather than on the science), and who is aware that software development involves rather more than mere coding”. The software engineer provides the technical expertise to modify software, for the purpose of

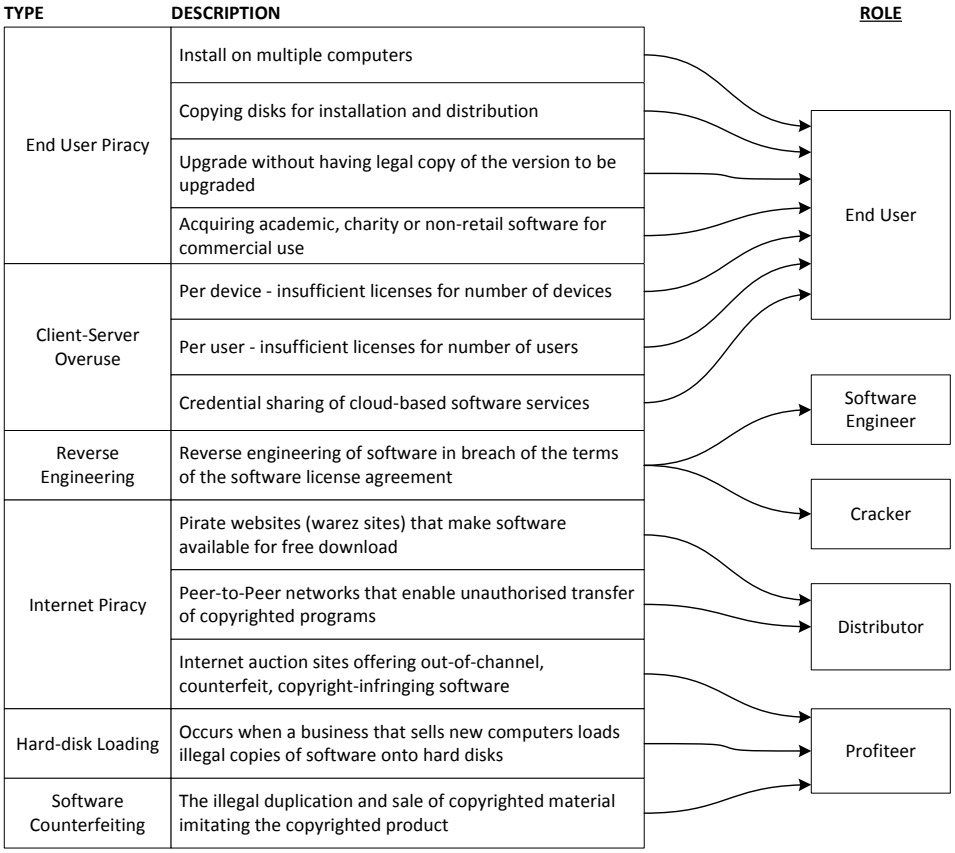


Figure 2.3: Associating Types of Piracy with Piracy Roles

understanding it and modifying it, for non-piracy purposes, against the terms of the EULA.

Cracker: crackers remove protections from copy protected software to allow piracy to occur, and have the most sophisticated, technical, and pivotal role in the piracy process. Their role is to remove copy protections from software using a variety of techniques, such as reverse engineering, executable code modification, and creating keygens (Goode & Cruise, 2006; Kammerstetter et al., 2012). Interestingly, crackers generally do not involve themselves in the other roles involved in piracy, having the primary motivation of the challenge of defeating copy protections (Goode & Cruise, 2006). Figure 2.4 on page 50 illustrates the role of the cracker

as one of removing the protection added to the software source code by the software vendor. Hence, once the legitimate software is obtained by the cracker, the software (and the software vendor) has no defence against the cracking reverse engineering or tampering processes.

All cracking activities are in breach of the rights holder license, as the "fair use" rights to reverse engineering are excluded in the software license. As most software requires the license terms to be accepted prior to installation (in this case for the purpose of preventing reverse engineering), the cracker will be in breach of the software license in these situations. Figure 2.4 on the following page has been adapted from Naumovich and Memon (2003), by changing output from a user perspective to an original/cracked software perspective.

Distributor: the Distributor is the facilitator of piracy through making the software available for download through the Internet. It is of interest that Distributors do not necessarily consume the pirated software product, but they provide a service enabling the piracy process: i.e. online file storage. Examples are businesses such as the former MegaUpload. Although this distribution is not a defined aspect of the TRIPS Agreement, it appears that each jurisdiction has made distribution of pirated software illegal, in order to close the gap created by the Internet.

Profiteer: the profiteer is a black market provider of pirated software that is illegally sold in breach of the TRIPS Agreement and jurisdictional laws, and as such denies the rights holder their revenues.

2.3.5 Taxonomy of Software Piracy by Role

Re-arranging the "Type of Software Piracy" from a perspective of the role shows who is participating in the piracy process and what their contribution is to the piracy process (Table 2.1 on page 51). This information is useful for understanding the types of threats

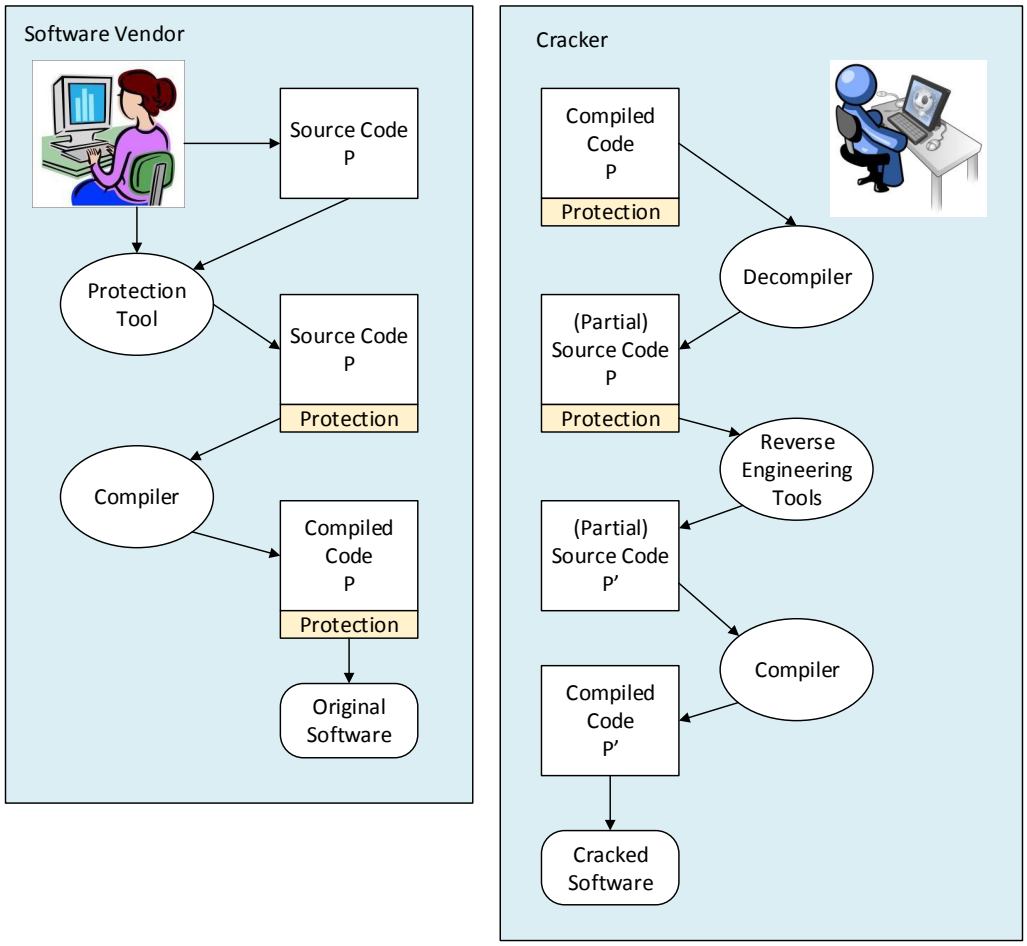


Figure 2.4: The Role of the Software Cracker (Adapted from Naumovich and Memon, 2003)

these roles represent as adversaries to copyright protection.

However, the relationships between the types of piracy is not defined by any of the sources, nor by the taxonomy presented in this section. Yet there is implied evidence, from the types of piracy, that relationships occur through the piracy role. For example: “End User” and “Distributor”, and “Cracker” and “Distributor”, are often associated with each other in the practice of piracy. Another example would be an “End User” who “cracks” an application and then distributes it to friends via a private BitTorrent.

The following sections build on the taxonomy of software piracy roles by taking the position that piracy is a process. It investigates piracy from differing perspectives to

Table 2.1: Taxonomy of Software Piracy by Role

Role	Type	Type SubCategories	References
End User	End User Piracy	Installation of software on multiple computers	(Mooers, 1977; Im & Van Epps, 1992)
		Copying disks for installation and not-for-profit distribution	(Suhler et al., 1986)
		Upgrade to a legal version of software without having legal copy of the version to be upgraded	(Business Software Alliance, 2014b)
		Acquiring academic, charity or non-retail software for commercial use	(Business Software Alliance, 2014b)
	Client-Server Overuse	Per device – insufficient licenses for number of devices	(Business Software Alliance, 2014b)
		Per user – insufficient licenses for number of users	(Business Software Alliance, 2014b)
		Credential sharing of cloud-based software services	(Business Software Alliance, 2014a)
Software Engineer	Reverse Engineering	Reverse engineering of software in breach of the terms of the software license agreement	(Electronic Frontier Foundation, 2015; Business Software Alliance, 2014b)
Cracker			
Distributor	Internet Piracy	Pirate websites (warez sites) that make software available for free download	(Goldman, 2004; Hétu, Morselli & Leman-Langlois, 2012)
		Peer-to-Peer networks that enable unauthorised transfer of copyrighted programs	(Kigerl, 2013; Xiaosong & Kai, 2009)
Profiteer	Internet Piracy (profit)	Internet auction sites offering out-of-channel, counterfeit, copyright-infringing software	(Palmer, 2014; Chaudhry et al., 2011)
	Hard-disk Loading	Occurs when a business that sells new computers loads illegal copies of software onto hard disks	(Business Software Alliance, 2014b)
	Software Counterfeiting	The illegal duplication and sale of copyrighted material imitating the copyrighted product	(Chaudhry et al., 2011; Yin-Leng, Wee Teck, May & Schubert Foo, 2010)

improve the understanding of the piracy process as well as the challenges surrounding piracy prevention. These ultimately present a piracy model that articulates a transitive relationship between role and prevention method, and demonstrates that piracy is clearly a process. Finally, the piracy process is further explored from a platform perspective, outlining the threats to each type of platform from the different types of piracy in the piracy process.

2.4 The Software Piracy Process

In this section the software piracy process is investigated, exploring the relationships between the type of piracy, the role of the actor, and the methods used in attempts to protect against piracy. A software piracy model is established, demonstrating the relationship between the protection method and the piracy actors. The types of piracy are then explored in the context of platforms, and the concept of the software piracy vulnerability lifecycle is introduced. The section concludes with discussion of related piracy issues and the documented link between piracy and malware.

2.4.1 Role-based Software Piracy Threat Model

A threat model by role can be derived from the taxonomy in Table 2.1 on the preceding page, and is shown in Figure 2.5 on the next page. The threat model visualises the relationship between the role and the type of piracy. This can be used to demonstrate how an adversary, who may fulfil multiple roles, creates a significant challenge for the rights holder to protect their copyrighted works. The threat model identifies “who” is undertaking the breach of copyright specifically, and relates that to “what” the breach of copyright is. It does not, however, say “how” a breach occurs, or which platforms are more vulnerable to a particular type of threat. These are new dimensions that the threat model cannot represent, and will be discussed in the following sections.

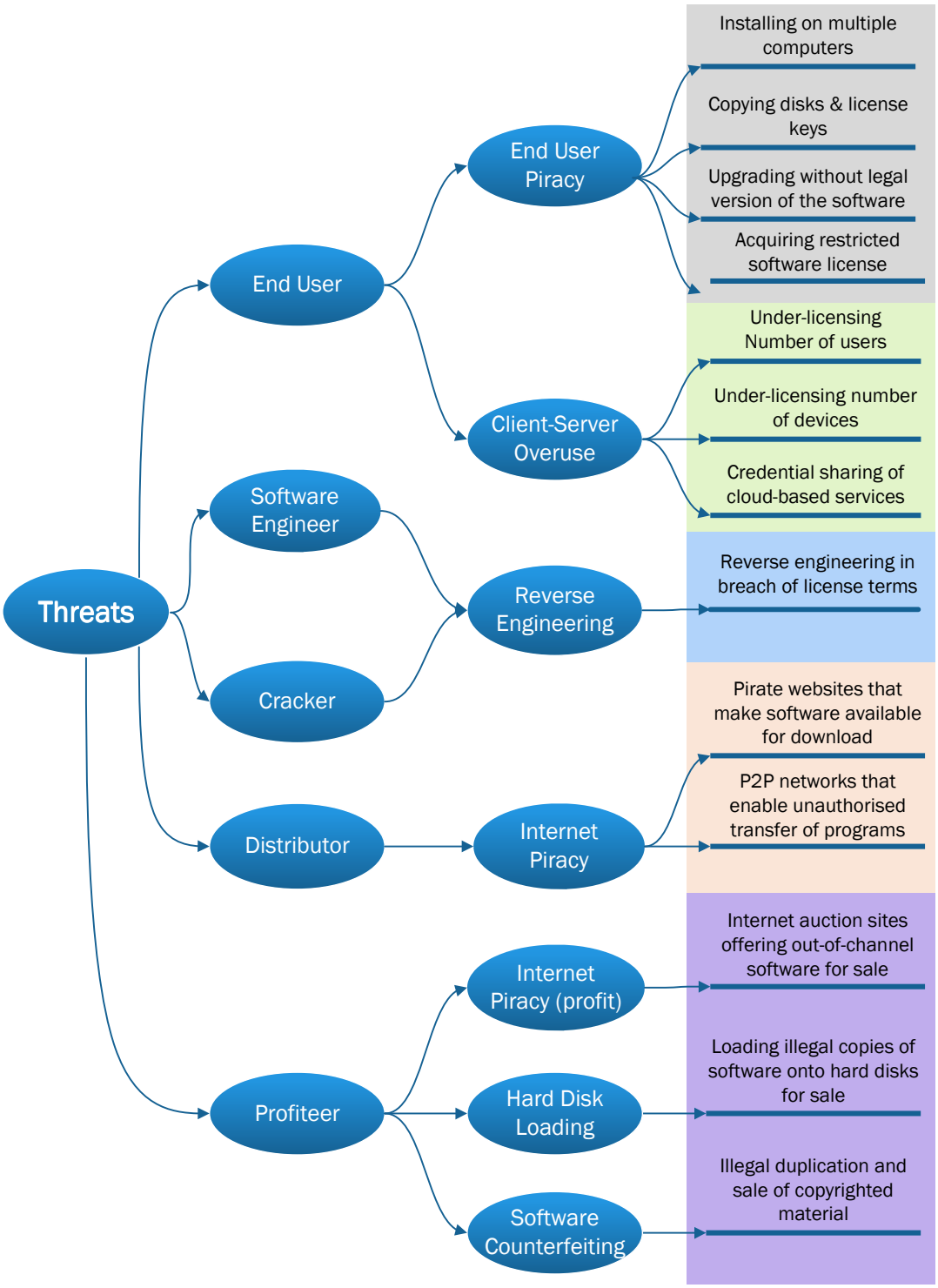


Figure 2.5: Software Piracy Threat model

2.4.2 Methods Employed to Reduce Software Piracy

The cracker piracy role exists specifically to overcome software piracy prevention mechanisms. To prevent piracy, vendors employ technical mechanisms that make it difficult for unauthorised actors to copy, install, and execute, their software. As vendors develop these new prevention methods, crackers respond in kind to defeat them.

This section first classifies the methods of protecting software copyright, focussing on the technical classes of prevention. It then re-categorises the types of software piracy, from Section 2.3.3 on page 44, into the technical classes of prevention, thereby demonstrating a transitive relationship, which will be discussed in the following Section.

At this point, it is useful to differentiate the terms "prevention" and "protection" as these are often used interchangeably in both research and public literature. *Oxford Dictionary* (2016) defines prevention as "related to precluding or hindering something"[6] or "related to preceding or anticipating something"[7], whereas protection is defined as "the action of protecting someone or something" or "a person or thing which protects someone or something". The "prevention" definition conveys a broader sense of frustrating a piracy actor through methodology and preparation, whilst the "protection" definition imparts the sense of specific safekeeping or safeguard of an entity from a specific type of threat. For the purposes of this thesis, "prevention" is defined as a class of methods that are proactively geared towards hindering piracy activity, whilst "protection" is defined as a sub-class of prevention, and is focussed on point technical mechanisms to defeat piracy actors.

Cronin (2002) proposes a taxonomy that defines three classes of methods used

[6] Prevention (I):
(i)The action of keeping from happening or making impossible an anticipated event or intended act;
(ii) The action of forestalling or frustrating a person in the execution of an action or plan;
(iii) The action of gaining advantage over a person by previous action;
(iv) Action intended to provide against an anticipated problem or danger; a defensive measure;

[7] Prevention (II):
(i) Action or occurrence before or in anticipation of the expected, appointed, or normal time; an anticipatory action, statement, etc

to prevent piracy – ethical, legal and technical. Ethical and legal piracy prevention methods are tactical imperatives to reduce piracy by encouraging users to "do the right thing" and instilling fear and uncertainty through risk of discovery and subsequent legal action (Wagner & Sanders, 2001; Suduc, Bizoi & Filip, 2009). As passive prevention methods, ethical and legal methods do not provide any capability to hinder piracy actors from breaching software copyright.

Protecting software from copyright breaches requires technical protection methods as shown in the taxonomy in Table 2.2 on the next page (adapted from Cronin's taxonomy[8]). Technical protection mechanisms have been identified as providing active protection from piracy actors. These include: (i) Copy Protection methods such as obfuscation, encryption, checking, tethering, observation, and watermarking (Conner, 1991; Naumovich & Memon, 2003; DeMarines, 2008); (ii) Authentication for cloud SaaS services and portals such as Steam, Adobe and Microsoft (Chaitanya, 2013; Ois, Sherif & Gamal, 2014); and (iii) Authorisation in the form of a License Key or License Server (Palmer, 2014).

Analysis of the taxonomy shows that there are gaps in respect to cloud-based services, and enhancements can be made to the Technical class to reflect these recent advances in applications, technology and business models. This includes the use of end user credentials as the means to protect Software-as-a-Service or Cloud-based software, and the use of license keys as a means of preventing piracy. Hence, revising the taxonomy to include Authentication and Authorisation Technical prevention methods increases the robustness of the taxonomy.

Table 2.3 on page 58 shows the types of piracy categorised into relevant protection methods as established in Table 2.2 on the next page. Each type of piracy was characterised into prevention methods belonging to the Technical Class from Table 2.2, and

[8] This taxonomy was adapted from Cronin's taxonomy by adding Authentication and Authorisation as additional technical methods of piracy prevention

Table 2.2: Taxonomy of Methods for Software Piracy Prevention (adapted from Cronin (2002))

Class	**Prevention Method**	**Sub-categories**	**Sub sub-categories**	
Ethical	Marketing, branding	Amnesty		
		Appeal		
		Shareware		
Legal	Statutes, Common Law	Copyright		
		Patents		
		License Agreements	Compulsory Audits	
			Duplication Restrictions	
Technical	Copy Protection	Obfuscation	Media Obfuscation	
			Code Obfuscation	Dynamic Obfuscation
				Static Obfuscation
		Encryption	Code Encryption	
			I/O Encryption	
		Simple Checking	Dongle	
			Registration	
			Guards	
		Tethering		
		Observation		
		Watermarking		
	Authentication	End-user credentials		
	Authorisation	License Keys		

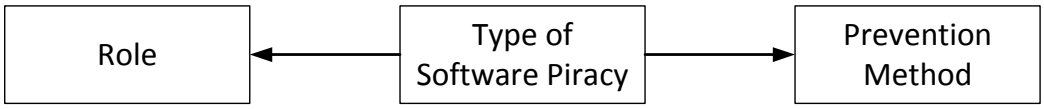


Figure 2.6: Software Piracy Transitive Relationship

in several cases, specific types of piracy could be characterised in both Authorisation and Copy Protection Prevention Methods. Some types of piracy were characterised as sub-categories of the Legal class and have been included for completeness of the model. The outcome of the categorisation determined that most of the sub-types of piracy are focussed on Authorisation and Copy Protection Prevention Methods, whilst Authentication is clearly a stand alone method solely used for services such as Software-as-a-Service and online software marketplaces.

2.4.3 Software Piracy Model

The Role-based Software Piracy Threat Model shown in the previous section was derived from the taxonomy of roles undertaking the types of piracy activities in Table 2.1 on page 51, whilst this section describes the technical Software Piracy Prevention Methods and categorisation of the types of piracy into these methods. Comparing these artefacts for common elements, it can be shown that there is a transitive relationship established between the Prevention Method and Software Piracy Role (abbreviated as Role) as shown in Figure 2.6.

The relationship between Prevention Method and Role can be demonstrated through a consolidated model of the piracy process as shown by the Software Piracy Model in Figure 2.7 on page 59. This model connects the Prevention Method (how software is protected) with Role-based Threats (who is defeating the Prevention Methods), and outlines the high level process by which this occurs.

Table 2.3: Types of Piracy Breach categorised into the Technical Class of Piracy Prevention Methods

	Classes of Piracy Prevention			
	Authorisation	**Copy Protection**	**Authentication**	**Legal**
Types of Breach	Installation of software on multiple computers	Installation of software on multiple computers	Credential Sharing	Upgrade to a legal version of software without having legal copy of the version to be upgraded
	Per device - insufficient licenses for number of devices	Copying disks for installation and not-for-profit distribution		Acquiring academic, charity or non-retail software for commercial use
	Per user - insufficient licenses for number of users	Reverse engineering of software in breach of the software license agreement terms agreed to		
	Pirate websites (warez sites) that make software available for free download	Pirate websites (warez sites) that make software available for free download		
	Peer to Peer networks that enable unauthorised transfer of copyrighted programs	Peer to Peer networks that enable unauthorised transfer of copyrighted programs		
	Internet auction sites offering out-of-channel, counterfeit, infringing copyright software	Internet auction sites offering out-of-channel, counterfeit, infringing copyright software		
	Occurs when a business that sells new computers loads illegal copies of software onto hard disks	Occurs when a business that sells new computers loads illegal copies of software onto hard disks		

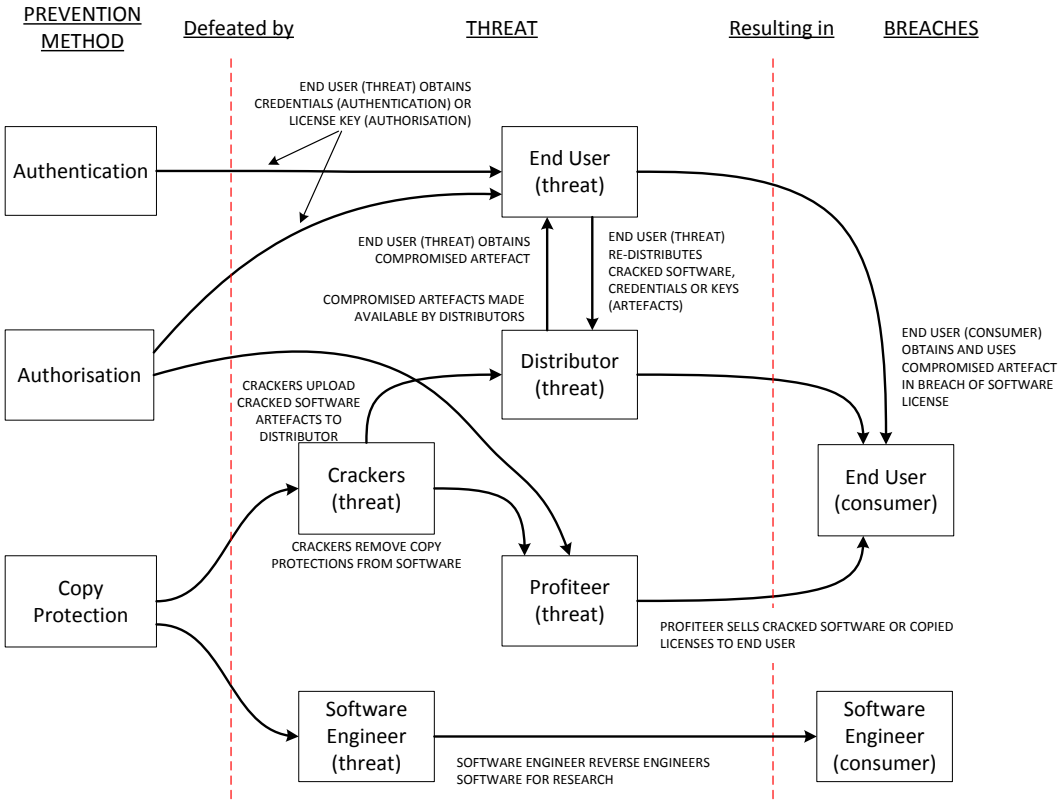


Figure 2.7: The Software Piracy Model

The Software Piracy Model uses the three prevention methods as the obstacles to be defeated into order to promulgate piracy. Prevention methods of Authentication, Authorisation and Copy Protection have separate paths into the Software Piracy model, representing that different prevention methods require different skillsets to defeat them. Threats are defined by the five roles of piracy, which have inter-relationships as the role of the software pirate changes for different stages of Software Piracy Model. The outcome of the Software Piracy Model is a breach of copyright or breach of license terms, and predominantly represent an economic impact on the rights holder. The End User (consumer) is the primary role for most piracy cases, where the end user is consuming pirated software or services, although not necessarily with the knowledge or realisation that they are using pirated software or services.

2.4.4 Defeating Prevention Methods

This section describes the Software Piracy Model in Figure 2.7 on the preceding page, and provides an outline of each of the elements identified in the model, and describes the relationship between the elements.

Authentication-based piracy is achieved through the sharing of credentials for online services or download portals (Business Software Alliance, 2014a). These credentials can be shared between End Users, where the initiator End User (threat) is considered the active threat, as they are the entity distributing the pirated software to other parties, as well as being the End User (consumer) and using the pirated software. Distribution can take several forms, including simple file sharing, online warez sites for public access, and peer-to-peer networks for semi-private sharing of license keys and files.

Authorisation-based piracy is based on the illegal use of license keys, where software license or activation keys can be distributed and used for non-commercial gain by End Users in a similar manner as credential sharing (Business Software Alliance, 2014a). In addition, license keys can also be used by Profiteers, who on-sell the illegal software either as packaged goods or pre-loaded on a computer, possibly without the End user (consumer) knowing that the software is illegal.

Copy protection represents the most significant challenge to piracy, requiring technically skilled Crackers to defeat the copy protection (Naumovich & Memon, 2003; Moon, 2009). After the cracker has successfully removed the copy protection mechanisms, the software vendor has no capability to regain control of the pirated software, and the "cracked" software can then be released for distribution, usually through warez sites or peer-to-peer networks, or to Profiteers. They are a key element of the piracy process and, as such, should be considered a primary adversary for software protection due to their skills and the fact that their motivation is the challenge, rather than the distribution of the software (Goode & Cruise, 2006).

2.4.5 Software Piracy Vulnerability Lifecycle

Figure 2.7 on page 59 provides a threat-oriented visualisation of how the piracy roles are cohesively interconnected to defeat piracy prevention methods. However, viewing the piracy problem from the platform perspective, as discussed in Section 2.2 on page 28, it can be shown that only certain types of platforms are vulnerable to certain types of piracy. Comparing each platform's vulnerabilities to the various types of piracy, it becomes apparent that the type of platform has relevance in respect to mitigation of piracy. Software, once released, has "States of Vulnerability" (SoV) relating to threat type. The more SoV that software traverses, the greater the vulnerability, and thus the increased risk that the software will be pirated (and copyright breached).

There are four SoV that have been modelled. Each SoV represents a point in the software lifecycle where the software becomes exposed to a type of piracy. Describing the relationship between these four SoV leads to the Software Piracy Vulnerability Lifecycle (SPVL), as shown in Figure 2.8. Therefore, the SoVs within the SPVL are defined as follows, and as shown by the Software Piracy Vulnerability Lifecycle Taxonomy in Table 2.4:

Release: where the software is available for purchase and can then be illegally upgraded, copied or counterfeited for resale, without any major technical impediment;

Installation: where the software needs to be installed, and technical expertise is required to overcome the piracy protection and prevention mechanisms through software cracking or reversing engineering;

Illegal Distribution: where the software is made available on the Internet for easy and convenient download and installation by the End User, with the software copy protection mechanisms removed;

Use: where the End User breaches the EULA, including renting/selling software, over-subscribing software use, and credential sharing of SaaS services.

Table 2.4: Software Piracy Lifecycle Taxonomy

State	Threat	Threat Sub-category	References
Release	Software Counterfeiting	Copying media & keys	(Suhler et al., 1986)
		Hard-disk loading	(Business Software Alliance, 2014a)
		Duplicating software package	(Suhler et al., 1986)
		Direct selling software product	(Chaudhry et al., 2011; Yin-Leng et al., 2010)
Installation	Software Cracking	Defeat license validation mechanisms	
		Defeat media protection mechanisms	
		Defeat software protection mechanisms	
		Subvert software authorisation	
	Reverse Engineering		(Electronic Frontier Foundation, 2015)
Illegal Distribution	Distribution of Illegal Software	Internet-based warez sites	(Goldman, 2004; Hétu et al., 2012)
		Internet counterfeit software sites	(Chaudhry et al., 2011)
		Peer-to-peer networks	(Kigerl, 2013; Xiaosong & Kai, 2009)
		Sharing digital media & keys	(Kigerl, 2013; Xiaosong & Kai, 2009)
Use	End User Breaches EULA	Sells or rents software	(Sony, 2016a)
		Breaches number of devices entitlements	(Business Software Alliance, 2017)
		Breaches concurrent user entitlement	(Business Software Alliance, 2017)
		Credential sharing	(Business Software Alliance, 2014a)
		Use of non-commercial license	(Business Software Alliance, 2017)

In Figure 2.8 on the following page, each of the primary platforms (Desktop, Gaming, Mobile and SaaS) enters the SPVL at a different point. Desktop platforms, which typically run Windows or Apple OSX, and mostly utilise COTS software, are immediate targets for piracy. These platforms enter at the Release state, because the COTS physical or digital media and keys are easily copied.

Console Gaming and Mobile platforms join the Desktop platform in the second SPVL state of exposure, "Installation". In addition to the Desktop platform software being a target on Release, Desktop platform software is also exposed to software cracking after Installation, whilst Console games, Consoles themselves, and the Mobile platform are targets for software cracking, modding, and jailbreaking.

Similarly, the "Illegal Distribution" state exposes all previous platforms to "Distribution of illegal software", and includes Mobile platforms – especially Android, due to its openness and ease of sharing Apps.

Finally, all previous platforms are joined by SaaS and Gaming Portals in the "Use" state of exposure, where EULAs for all and any platforms may be breached and credential sharing occurs.

Hence, the model shows that the Desktop Platform has the highest exposure to piracy and is the hardest platform for which to prevent piracy. On the other hand, SaaS services and Gaming Portals have the least exposure to piracy, as credentials form the basis of entitlement. However, many SaaS services are simply Digital Distribution Portals for Desktop platforms, and distribute installable software – in which case the SPVL state is effectively at the "Release" exposure state. It can be concluded that the vendors who control the authentication, authorisation and the client endpoint, have better control over their software distribution and less exposure to piracy.

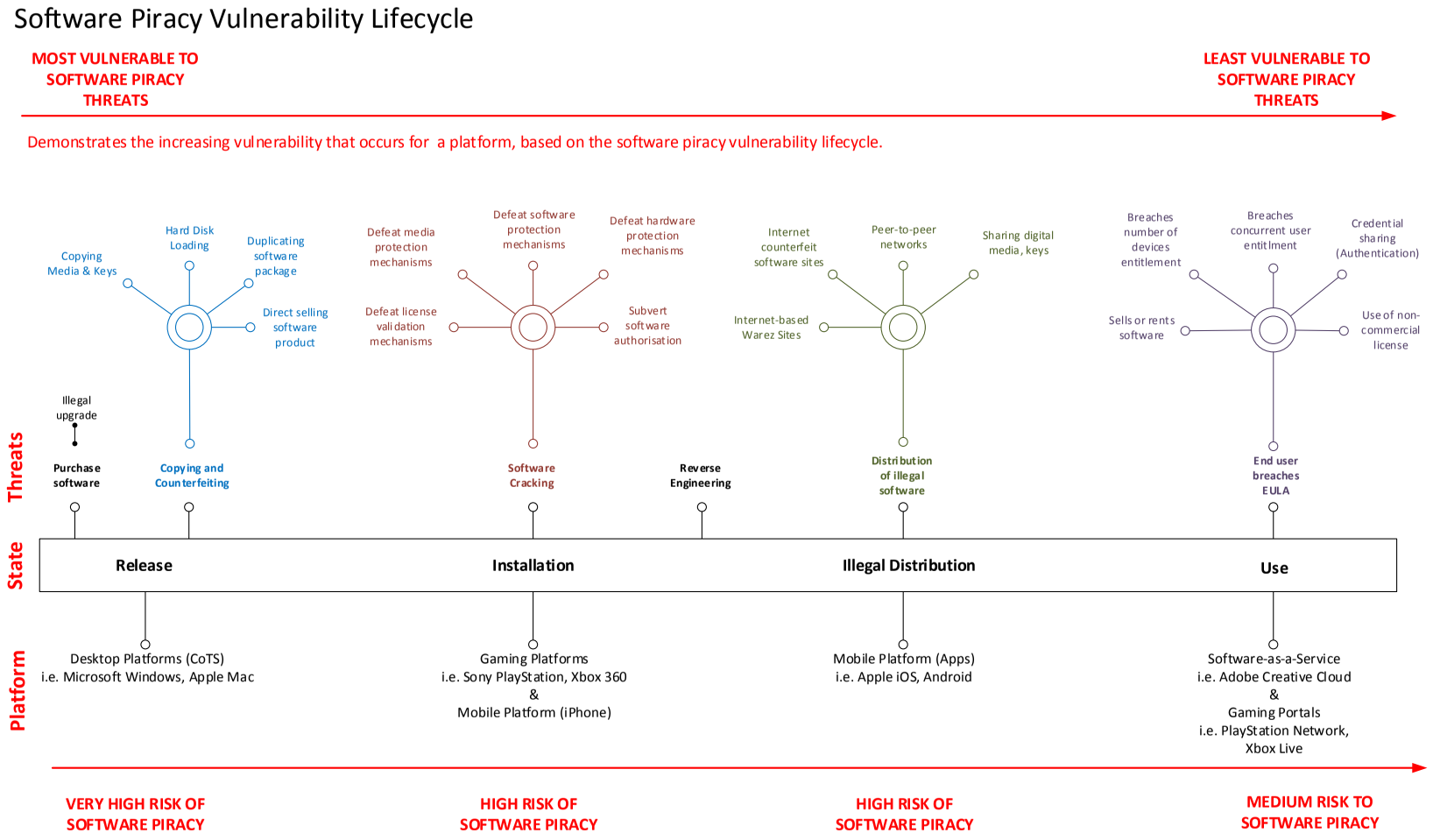


Figure 2.8: Software Piracy Vulnerability Lifecycle

2.4.6 Platform Risk

Quantifying the number of SoV per platform shows the level of comparative risk that each platform has to the piracy process (Table 2.5). Although the objective of this study is not to quantify the actual risk, where possible the Global Unmonetised Demand for software is provided (shown by the Impact column) to give an indication of the value lost through piracy across the different platforms. However, aside from the BSA report for piracy, quantifying loss in respect to desktop operating systems and productivity software, there seems to be little research quantifying piracy losses across other platforms. However, one report, from Arxan Technologies (2015), finds that the Games Global Unmonetised Demand (loss) is estimated at US$74 billion across all game platforms (Desktop, console, closed mobile and open mobile). Adding this cost estimate for games to the BSA cost estimate for Operating System (OS) and productivity software, the impact from piracy goes from US$58 billion to US$132 billion, a 228% increase[9]. This demonstrates that the severity of the total piracy issue is considerably larger than the BSA analysis suggests. The quantities presented in Table 2.5 have been derived from the position of the SoV shown in Figure 2.8 on the previous page. This provides an indication of the number of SoV per platform, and the contribution to the Global Unmonetised Demand cost.

2.4.7 Related Issues

The responsibility for the detection of breaches of copyright and enforcement of software copyright and license terms lies with the software vendor. Although the copy protection rights may be enacted into law, as per the signatories to the TRIPS Agreement, any legal process is undertaken by the vendor, who may use breach of statute or a commercial breach of terms as the trigger mechanism for legal action. Alternatively, the BSA

[9] This figure excludes: (i) piracy of non-game software on mobile devices; (ii) SaaS related piracy.

Table 2.5: Threats by State of Vulnerability

Platform	**Release**	**Installation**	**Illegal Distribution**	**Use**	**Total SoV**	**Impact**
Desktop OS	4	5	4	5	**18**	US$58B (2013)
Closed Platforms (Consoles & iPhone)		5	4	5	**14**	Games $74B (2015)
Open Platforms (Android)			4	5	**9**	
SaaS				5	**5**	unknown

provides compliance and enforcement services in over 60 countries, investigating 15,000 reported cases of piracy in 2012 alone. The BSA services include enforcement, government engagement, public awareness campaigns, techniques to detect/disrupt piracy, and compliance tools and services (Business Software Alliance, 2015). The FBI supports this position of enforcement, stating that preventing intellectual property theft is a priority (Federal Bureau of Investigation, 2015). The FBI has initiatives such as the FBI Anti-Piracy Warning Seal, which is intended to remind users about the serious consequences of pirating copyrighted works.

However, when considering piracy from a geographic perspective, enforcement is clearly a challenge. Piracy rates vary based on geographic national boundaries as a result of economic differentials between the software cost and affordability in the country (Asongu, 2014; Moores & Dhillon, 2000). The BSA Compliance Gap report (Business Software Alliance, 2014a) supports this analysis, revealing piracy rates of 80%-90% for third world countries, compared to 20%-40% in first world countries. Furthermore, countries such as China, with 77% piracy rate, have social and cultural values that espouse piracy as the smart and pragmatic choice, from an individual's perspective (Rapoza, 2012).

2.4.8 Relationship between Malware and Pirated Software

Pirated software has long been a vector for the distribution of malware, which is often embedded in the downloaded or distributed software (Gantz et al., 2015). Kammerstetter et al. (2012) finds, after analysing 43,900 download links and more than 23,100 cracks and keygens, that over 50% of pirated (desktop) software is infected by malware.

From a smartphone perspective, online application stores are also a means of distributing malware infected software[10]. Infected Apps, mimicking genuine Apps, can easily be uploaded, and are downloaded by unsuspecting end users ("Android marketplace hit by malware", 2011; F-secure, 2014).

Gantz et al. (2015) concludes that reducing piracy will have a strong correlation with a reduction in malware distribution: "For enterprises, governments, and consumers, the obvious implication is that one way to lower cybersecurity risks is to reduce the use of unlicensed software".

2.4.9 Summary of Software Piracy Review

This section first shows that there is a transitive relationship between the type of software piracy, the piracy role, and the prevention methods the publishers use to reduce piracy of their software. A software piracy model (see Figure 2.7 on page 59) is established through these findings, and reveals that the common factor in the software piracy process is the End User role, despite the involvement of the other roles in the process. These finding apply regardless of whether the End User actor is aware of the piracy or not. For example, an End User may be using an unlicensed application in the work environment, or may have received a copy of software without knowing it was pirated software.

Building on this, the section then explores the SPVL (see Figure 2.8 on page 64), and demonstrates how vulnerable software is to piracy throughout its lifecycle. It outlines

[10] Excluding closed ecosystems such as the Apple App store, where developers must undertake an App review process to ensure their App complies with Apple policies

software vulnerability in relation to platforms, and how each platform enters the SPVL based on their software delivery model. The SPVL outlines the SoV for software from "Release" state, to the "Use" state, and demonstrates that the "Use" state is the least vulnerable to piracy threats. As an example, migrating to "Use" state business models is a driver seen in the cloud computing Software-as-a-Service industry, where publishers have developed a business model for consumers to purchase software legitimately based on use.

The literature reveals that existing methods to protect software copyright are ineffective, and that the combative state between publishers and piracy actors to protect software will likely be on-going. There is little research on estimated losses due to piracy across platforms other than desktop OS. However, the estimated impact, from current literature, places this cumulatively at US$132 billion per year, with cross-platform losses higher than desktop OS. Hence, there is a strong motivation to solve this problem.

The key learning is that to protect software copyright across multiple platforms, the consumer (end user) is the single common factor. If there is to be a successful method to prevent software piracy, it will need to be directed at providing some form of SLV method for the consumer at the time of use.

The following section is the second part of this Chapter and explores cryptocurrency and blockchain technology to identify the feasibility of applying the blockchain as a method that may help overcome the challenges around protecting software copyright. Cryptocurrency and blockchain technologies present an interesting opportunity, and a possible solution, to the software piracy challenges identified in this Section. The following Section introduces cryptocurrency and blockchain ecosystems, and explores the uses for cryptocurrency and blockchain technologies beyond the monetary transactional precepts. It introduces the cryptocurrency model, which categorises the different implementations of the cryptocurrency blockchain for use by applications.

2.5 Blockchain Technology

2.5.1 Introduction

In the previous section, the software piracy process was reviewed, and the need for a user-based SLV method to prevent piracy and protect software copyright was established.

This section reviews the literature encompassing cryptocurrency and blockchain technology, to identify the feasibility of using blockchain technology, both to create a method to address the software piracy prevention and provenance issues, to provide controls through which software creators might protect their copyright. It explores the origin of cryptocurrency, and cryptocurrency's uses for applications outside the monetary transaction domain.

2.5.2 Cryptocurrency Primer

Developed by Nakamoto (2008) and first introduced with the creation of Bitcoin, cryptocurrencies are a new form of virtual currency. A cryptocurrency is a distributed peer-to-peer electronic cash system and is the first technology to successfully overcome the requirement for a centralised party to validate transactions. The cryptocurrency and blockchain architecture provides several blended features, which include: (i) cryptographic validation for all transactions; (ii) decentralised money; (iii) the minting of bitcoins; and (iv) transaction processing functions, all stored on public ledgers within a quasi-anonymous framework (Brikman, 2014). Cryptocurrencies use public-key cryptography to validate transactions between all participants, and digital signatures to ensure transactional integrity and non-repudiation (Peteanu, 2014). The cryptographic mechanisms used by cryptocurrencies provide for strong confidentiality, data integrity, and non-repudiation services that are in use by business, government and military organisations globally. In a cryptocurrency ecosystem, the public key can be considered as

the participant's account number whilst the private key represents the participant's ownership credentials. All participants have digital wallets, which are used to store private keys, as well as digital signatures that represent cryptocurrency entitlements (bitcoins) that the participants own. Wallets can be stored privately, or online on websites or in exchanges, depending on the requirements of the participant. Needing to be resilient to threats and attacks, as well as being a stable and liquid currency, cryptocurrencies have yet to prove their robustness both technologically (Bradbury, 2013; Courtois, 2014) and economically (Hanley, 2013; Plassaras, 2013).

However, as developers seek to use cryptocurrencies in more practical applications, it is the cryptocurrency architecture, the blockchain, that is the point of development of new cryptocurrency based applications (Duivestein & Savalle, 2014; Prisco, 2015b; Bass, Bault, Baum, Channell & Englander, 2014).

2.5.3 Cryptocurrency Economics

The objective of a cryptocurrency is to overcome the necessity of a centralised "trusted authority" (Nakamoto, 2008), and thus remove or significantly reduce transaction fees associated with transactions, such as those incurred with commercial banking transactions. As a peer-to-peer distributed technology, cryptocurrencies historically rely on a network of low-cost computers, running software that performs primary functions within the cryptocurrency ecosystem. The term ecosystem, used in this context is not the usual portmanteau of "ecological system", but instead evokes a new portmanteau: "economic system" (Herbert & Litchfield, 2015). That is, the cryptocurrency creates an economic system that incentivises the participants to invest in resources (e.g. purchase computers for running cryptocurrency software), and receive a return on their investment (transaction fees and bitcoins). The computers running this software are "Miners", which create bitcoins (Brikman, 2014), validate bitcoin

transactions, and maintain the integrity of the blockchain public ledger. Miners are rewarded for their investment in running the bitcoin software by creating the bitcoins themselves. Miners may also (but not always) receive a transaction fee for validating bitcoin transactions. For example, participants in the Ripple (Buterin, 2013) and GridCoin (Halford, 2014) cryptocurrencies run transaction validation software, on a voluntary, non-profit basis, in return for being allowed to provide compute, and storage resources to run the mining software.

Figure 2.9 on the following page provides an overview of the cryptocurrency economic system, showing how the actors interact with the blockchain. Using a generic cryptocurrency as an example, the figure shows that the miner runs software that undertakes three primary functions of minting coins, validating transactions, and managing the blockchain database, which is stored locally on the miner's computer. Each actor has its own wallet, located on its computer, which is used to store the digital signatures representing bitcoins that each actor owns. Each actor also has software that interacts with the Bitcoin ecosystem.

When a customer purchases goods from a merchant using bitcoin, the software on the customer's computer communicates the transaction details to a pool of the many cryptocurrency mining nodes, which then jointly validate the transaction within a set period of time[11].

As developers seek to use cryptocurrencies in more practical applications, it is the blockchain that is the point of interest of new distributed applications (Bass et al., 2014; Duivestein & Savalle, 2014; Prisco, 2015a). The blockchain itself will be discussed in further depth in section 2.5.6 on page 74.

[11] This is 10 minutes for the Bitcoin protocol. Other cryptocurrencies vary this transaction validation time parameter to improve certainty.

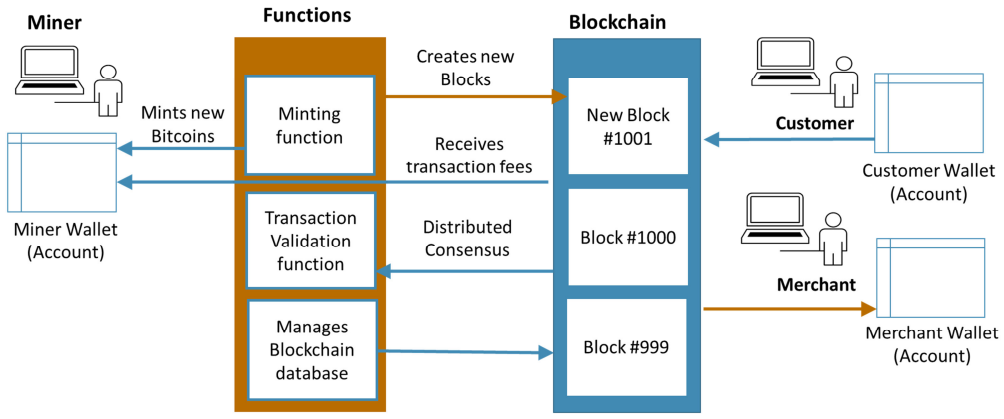


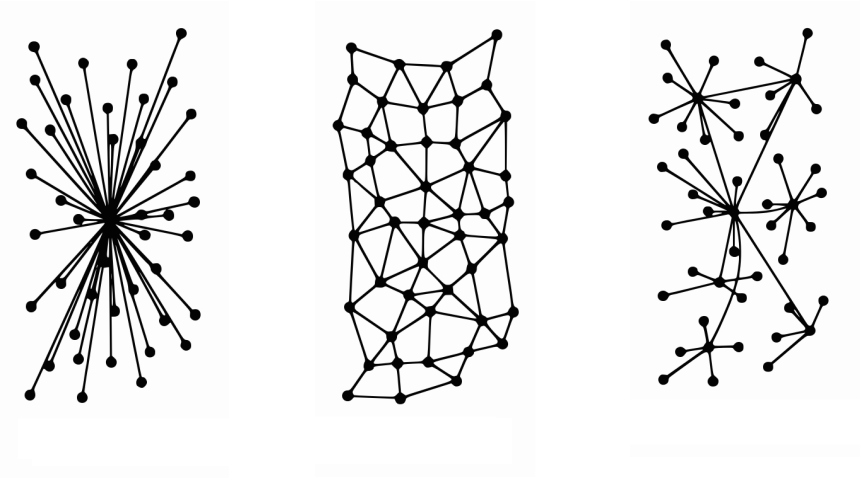
Figure 2.9: Cryptocurrency Economic System

2.5.4 Centricity

In this section, cryptocurrency centricity is discussed as the fundamental characteristic (Herbert & Stabauer, 2015) that allows the cryptocurrency to function over a public domain such as the Internet.

Centricity is the characteristic that makes the cryptocurrency ecosystem of an attractive proposition to stakeholders. Baran (1964) identifies three types of network centricity: centralised, decentralised, and distributed, as shown in Figure 2.10. All of these are used in currency and cryptocurrency systems (Herbert & Stabauer, 2015). The global fiat currency banking system is centralised as shown in Figure 2.10 (a), to validate all currency transactions. Most cryptocurrencies are distributed, as shown in Figure 2.10 (b), where there is no central point of failure and no master node required for the transaction validation process. A few cryptocurrencies, such as Ripple (Buterin, 2013), are decentralised, with a master node for transaction validation as shown in Figure 2.10 (c).

Combining distributed centricity with the Internet as a medium of transport results in an artefact that is accessible from anywhere, creating a new reference characteristic of ubiquity.



(a) Centralised (b) Distributed (c) Decentralised

Figure 2.10: Centricity (Baran, 1964)

2.5.5 Transactions

Bitcoin transactions are defined as a message between participants and consists of three segments (Brikman, 2014): (i) Signature – the originator's digital signature, signed with the originator's private key, so that other Bitcoin nodes can verify that the message came from the originating participant; (ii) Inputs – a list of the signatures of transactions already in the ledger where the originator was the recipient of bitcoins, and the input bitcoins are the funds the originator used in the transaction; (iii) Outputs – a list of how the funds in the inputs are to be distributed. All the fund inputs must be redistributed as outputs, so the originator will pay the recipient the required amount, and the remainder is returned as change.

A transaction must have exactly the same number of bitcoins in input and output lists. Hence, if user U1 has 10 bitcoins and wants to send 2 bitcoins to user U2, then the transaction will result in U1 receiving 8 bitcoins and U2 receiving 2 bitcoins, as shown in Figure 2.11 on the following page. That is, 8 bitcoins are reassigned to U1. Note that if the transaction occurs with an unequal result, then the transaction fails to

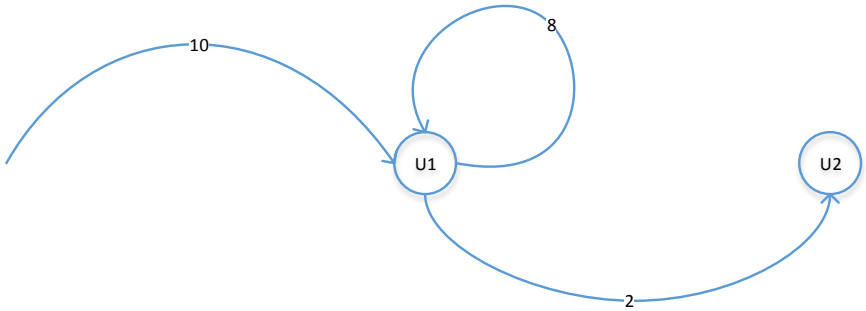


Figure 2.11: Transaction Input and Output
(Herbert & Litchfield, 2015)

complete with a valid or consistent state. This is shown by the following, where U1 represents User1, the sender of the 2 bitcoins, and U2 represents User2, the recipient of the 2 bitcoins.

U1.input(U1, 10)

U1.output(U2, 2)

U1.output(U1, 8)

2.5.6 The Blockchain

In this section, the blockchain mechanisms are explored further. It explains how blocks are generated, how the chain of blocks is created, and how the transactions stored within the block are validated.

Blocks are created at regular intervals – for example every 2.5 minutes or every 10 minutes – and each block contains the following properties:

1. Transactions (T) sent between participants
2. Proof of Work (P_w) – a unique digest created when a new block is discovered, and

3. Reference (R) to the digest of the previous block.

Thus the block (B) is represented as

$$\exists B : B(T, P_w, R)$$

Wood (2016) defines a block as a collection of transactions that are chained together with a cryptographic hash that operates as a reference mechanism[12]. A block functions "as a journal, recording a series of transactions together with the previous block and an identifier for the final state" (2015, p 2) although, due to size constraints, the journal does not record the final state. The key to successful mining is reliant upon a cryptographically secure proof.

Miner nodes are continuously computing hash values to solve a problem, from which bitcoins are subsequently rewarded to the miner. Bentov, Gabizon and Mizrahi (2014) define P_w as the means by which confidence is given to a bitcoin, as a consequence of it being difficult to replicate in respect to the computation difficulty. The generated hash values (the values of P) are then inserted into a block.

The purpose of this is to create what is called consensus, which is the process for agreement that the block and the transactions held within are valid. Consensus is reached when a majority of nodes on a network that have the capability of mining bitcoins have voted to accept the validity of a block of transactions. Thus, when a transaction occurs, the event is broadcast on the cryptocurrency network for verification by multiple Miner nodes. This verification process is designed to prevent double spending of the bitcoin. When each node has received the transaction, the node tests the validity of the transaction. The more nodes that accept the transaction, the less likely it is that the transaction represents a double spend.

A transaction may end with one of three states:

[12] The cryptographic hash is unique for each block, and each block has a reference to the previous block using the hash. Hence, the chain of blocks can be easily identified and verified.

1. Confirmation of the successful completion of a transaction, which does not occur until 6 consecutive blocks[13] of transactions have been validated[14], thereby fully confirming the transaction within the block is valid.
2. Incomplete transaction, for example if the immediate responding nodes reject the transaction. Otherwise the transaction has timed out after a period, typically 60 minutes for Bitcoin specifically.
3. Unverified transactions[15] are placed in an unverified transaction bucket, and will be inserted into the next block once it is created. Any node in the Bitcoin network can put several unverified transactions into a block and send it out to the rest of the network as the proposed next block in the chain.

The current new block then has a new value P_w^1 inserted as its Proof of Work (Figure 2.12 on the next page). When a block is verified, the value P_w^1 is inserted into the next block. It is this sequence of $\{ P_w^1, P_w^2, \ldots, P_w^n\}$ that defines the unique blockchain and its associated record in the ledger.

2.5.7 Summary of Blockchain Technology

This section has provided a brief overview of how cryptocurrency and blockchain technologies function to create a distributed consensus ecosystem.

The following sections explore some alternative applications for cryptocurrencies and the blockchain, finishing with an in-depth exploration of how the blockchain could be used to prevent piracy.

[13] The Bitcoin ecosystem requires 6 subsequent blocks to be completed before a transaction in a block is considered valid. This gives the Miner nodes the opportunity to arrive at consensus for all transactions within a block

[14] The Bitcoin ecosystem requires 5 Miner nodes to validate transactions

[15] Unverified transactions are transactions that are not entered into the blockchain yet, awaiting the next block to be created

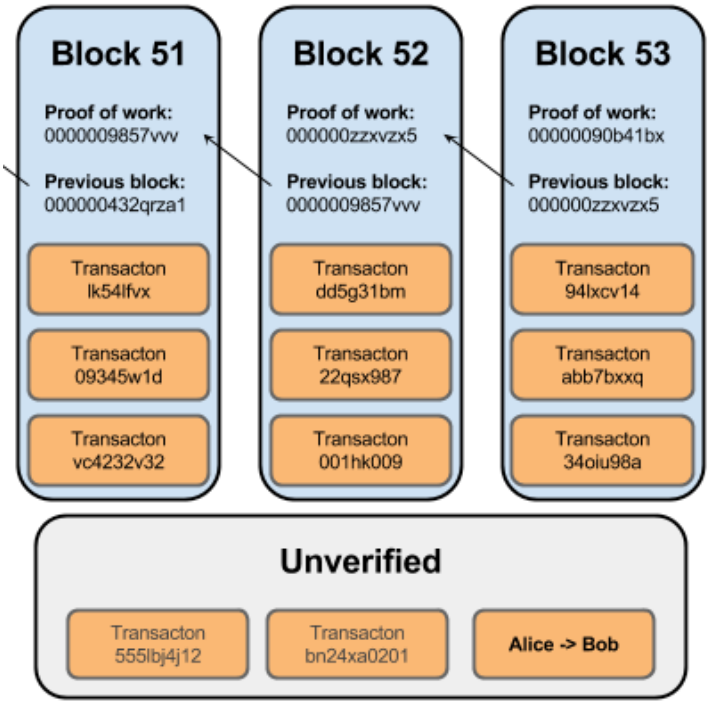


Figure 2.12: Example of the Blockchain (Brikman, 2014)

2.6 Alternative Applications for Blockchain

The distributed nature of the blockchain, combined with the ability to verify transactions, has led to the development of second generation blockchain protocols. These blockchain protocols have the capability to perform new functions, supporting innovative on-blockchain services, such as storing data and new scripting capabilities (Bradbury, 2014; Buterin, 2017). This section introduces the different methods in which cryptocurrencies and the blockchain can be implemented to provide a non-financial output, application service, or function.

Herbert and Stabauer (2015) proposes a model, categorising cryptocurrencies and the different implementations of the cryptocurrency blockchain for use by applications. This model, shown in Figure 2.13 on page 79, differentiates several types of blockchain applications that move beyond solely providing cryptocurrency transaction functions.

The types of cryptocurrency and blockchain applications defined in the model include the following:

- Transaction-only Cryptocurrency (TOC)
- Native Blockchain Application (NBA)
- External Blockchain Application (EBA)
- Transaction and Application Platform (TAP)
- Regulated Virtual Currency (RVC)

These types of cryptocurrency and blockchain applications are discussed in the following sections. The "Types of Currency Model" also provides abbreviations for the "type" of cryptocurrency, based on the term "Virtual Currency Scheme" (VCS) defined by the European Central Bank (European Central Bank, 2015). This research has used the terminology in the above list for clarity, rather than the VCS1 through VCS4 types described in the model.

2.6.1 Transaction-only Cryptocurrency

Applications that utilise an existing transaction-only cryptocurrency for its own purposes are TOC applications, also known as Appcoins. Developers can create custom applications that store custom data in the blockchain data field, to create an immutable record. Appcoins utilise the host blockchain ecosystem for the benefits of having an established ecosystem, and avoiding the overheads associated with running a cryptocurrency.

ColoredCoins[16] are Appcoins that utilises the Bitcoin blockchain (Assia, Buterin, Hakim & Rosenfeld, 2014), and supports a limited scripting language, which can be used to store metadata on the blockchain. By attaching metadata to Bitcoin transactions, ColoredCoins leverages the Bitcoin infrastructure for issuing and trading immutable

[16] http://www.coloredcoins.org

Legal Status	Regulated	Unregulated								Regulated
Centricity	Centralised					Decentralised/Distributed				
Format	Physical or Digital	Physical	Digital							
Control and Issuance	Government	Private				Network/Private				Government
Validating System	Bank	Private				Algorithm (PoW, PoS etc)				
Source	Closed Source					Mostly Open Source				
Purpose	Transaction Only						Transaction & Application		Transaction & Application Platform	?
Function Integration	Not Applicable						Embedded	External	Embedded	?
Type	**FIAT**	**LOCAL**	**ECB 1**	**ECB 2**	**ECB 3**	**VCS 1**	**VCS 2**	**VCS 3**	**VCS 4**	**RVC**
Definition	**Fiat**	**Local Currency**	**Closed**	**Uni-directional**	**Bi-directional**	**Transaction only Crypto-currency**	**Native Blockchain Application**	**External Blockchain Application**	**Application Platform**	**Regulated Virtual Currency**
Examples	USD/EUR E-Money	Community Coins Detroit Scrip Freigeld	Warcraft Gold	Facebook credits Frequent Flyer Miles	Linden Dollars	Bitcoin, Peercoin, Ripplecoin, Solarcoin, Zerocash, Primecoin, Colored Coins	NXT, Namecoin, Maidsafe	Omni, Sidechains, Counterparty, Gridcoin	"Turing Complete Platform" "Bitcoin 2.0" Ethereum, Zerocash	RScoin (concept) Future

Figure 2.13: Types of Currency Model

digital assets that can represent real world value. The metadata can be used as tokens, to represent real world assets or represent transactions linked to real world assets. ColoredCoins allows users to "colour" a token to represent a specific asset such as a car, home, boat, commodity, shares, or bonds (Rosenfeld, 2012).

Fortin (2011) demonstrates how a TOC can be leveraged to establish provenance of a software license by associating the real asset (the software license) with a digital asset (a cryptocurrency coin) and an authorising bitcoin address, called the Master Bitcoin Model (MBM). This combination establishes the ownership of the software through the immutability of the blockchain. Whoever has the bitcoin from that specific address in their wallet has license rights, and the bitcoin can change ownership simply by sending it to another wallet, as in any standard cryptocurrency transaction.

2.6.2 Native Blockchain Application

NBAs provide additional application functionality built into the cryptocurrency or blockchain. Although most cryptocurrencies rely on financial oriented transactions, this type does not require an application to have a monetary purpose. The NBA functionality provides the capability for design and engineering of customised blockchain applications that meet specific use-case requirements. Custom applications are clearly being used within the blockchain industry and community, and blockchains are being advocated globally through dedicated conferences[17] and academic journals[18]. NBA examples include:

- MaidSafe[19] is a distributed application that provides a fully decentralised platform on which application developers can build distributed applications where data

[17] BraveNewCoin Events Calendar https://bravenewcoin.com/industry-resources/events-calendar/bitcoin-and-blockchain-events/

[18] Elsevier https://www.journals.elsevier.com/future-generation-computer-systems/call-for-papers/special-issue-on-cryptocurrency-and-blockchain-technology

[19] http://www.maidsafe.net

storage is automatically distributed. The MaidSafe ecosystem runs on the Secure Access For Everyone (SAFE) network, which is an open source, decentralised data storage and communication platform. Nodes connected to the SAFE network collectively store data for all MaidSafe users, and provides public and private cloud data storage and processing (Irvine, 2014).

- NameCoin[20] is an experimental open-source technology that improves decentralisation, security, censorship resistance, privacy, and speed of certain components of the Internet infrastructure such as DNS and identities.
- Acronis (2016) is undertaking research and development on blockchain-based technology to ensure data authenticity, privacy and control.

2.6.3 External Blockchain Application

Similarly to NBAs, EBAs also provide additional application functionality to the cryptocurrency or blockchain ecosystem, but do so through an external interface to the blockchain. These applications process the blockchain natively, and are programmed to interact accordingly with the blockchain. EBAs are alternative currencies with their own bitcoins, holding their own value, but sitting on the host cryptocurrency ecosystem, (typically Bitcoin).

However, EBAs are centralised in nature (they run as a service and require their own server to function), and are under the control of a third party developer. Examples of EBAs include the following:

- Counterparty[21], which uses the Bitcoin core network, enables anyone to write specific digital agreements and execute them on the Bitcoin blockchain (Winters, 2014).

[20] http://namecoin.org
[21] https://counterparty.io/

- Gridcoin[22] is a blockchain protocol developed for the Berkley Open Infrastructure for Network Computing (BOINC) hosted network. BOINC provides distributed computing processes such as SETI@Home (Search for Extraterrestrial Intelligence), and Poem@Home protein folding modelling.
- Omni[23], which is a platform for creating and trading custom digital assets and currencies, utilises the Bitcoin core network.

2.6.4 Transaction and Application Platform

A TAP is defined as a cryptocurrency blockchain that provides an embedded Turing complete contract language, and which also enables Distributed Applications (DApps) to run across the blockchain ecosystem (Herbert & Stabauer, 2015). This functionality is distinct, in that the Turing completeness will allow conditions and loops to be implemented: something that other cryptocurrencies do not support by design. Each transaction may have a Smart Contract associated with it to provide transaction level rules, based on any internal or external factors. For example, an Internet of Things vending machine can negotiate a contract with suppliers for replacement orders, with parameters including pricing, delivery schedules, loading the machine, acceptance and payment. The Ethereum cryptocurrency is the sole example of a Transaction and Application Platform.

Ethereum aims to overcome many of the threats and issues of first generation cryptocurrencies, including value-blindness, lack of state, and blockchain blindness (Buterin, 2017). The objective is to allow creation of distributed applications that automate intelligent processes between participants, including devices, which will evolve additional markets such as Machine to Machine (M2M), and Machine to Consumer (M2C). Duivestein and Savalle (2014) suggests that blending The Internet of Things

[22] http://gridcoin.us/
[23] http://www.omnilayer.org

and DApps enables all the types of objects to be connected to the Internet to interact with the blockchain. Examples are given of vending machines being able to re-order stock directly without the requirement of any fulfilment intermediaries, and the advent of Business to Machine (B2M), Machine to Business (M2B), Consumer to Machine (C2M), M2C, and M2M markets (Duivestein & Savalle, 2014). The following list outlines some of the possibilities for the distributed blockchain applications (Ledra Capital, 2014; Buterin, 2017).

- Financial Instruments and Records, such as currency, equities, mortgage records, and crowdfunding
- Public Records, such as land titles, vehicles registries, criminal records, birth certificates, and voting
- Private Records such as contracts, signatures, wills, trusts, and escrows
- Other Semi-Public Records, such as degrees, grades, medical records, accounting records, and genome data
- Physical Asset Keys such as home, time share, hotel, car, and rental cars
- Intangibles such as coupons, vouchers, reservations, movie tickets, copyrights, trademarks, and DRM.

2.6.5 Regulated Virtual Currency

There are distinct advantages of cryptocurrency technology in a government sense, where transactions are taxed automatically at the time of transaction, regardless of the global location of the participants, and taxing of micro-transactions are easily supported. The government could insert various algorithms in a government controlled cryptocurrency to enable and automate many types of financial, health and social support processes. Danezis and Meiklejohn (2015) proposes RSCoin, which provides a centralised monetary policy combined with a distributed validating system for prevention

of double-spending. These characteristics make it suitable for government use, where ownership of the cryptocurrency is essential, as indicated by Herbert and Stabauer (2015).

Furthermore, World Economic Forum (2015) predicts that tax will be collected for the first time by a government via the blockchain by 2023, whilst by 2027, 10% of global GDP will be stored on blockchain technology. The World Economic Forum states that "Economic and monetary management will be overhauled by new systems anchored in digital currencies and the blockchain." Similarly, Moy (2014), former Director of the U.S. Mint, expresses how cryptocurrencies are evolutionary and government interest is expected to grow.

2.6.6 Summary of Alternative Applications for Blockchain

This section has discussed several types of uses for blockchain technology, based on the types of currency model, which is defined through identifying how the application makes use of the cryptocurrency and blockchain. For the most part, applications that use the cryptocurrency for monetary transactions use an existing cryptocurrency. Applications that store metadata on a blockchain also use an existing cryptocurrency. However, applications that have special requirements in respect to functionality or resources are created by developers as a NBA, where the entire blockchain is tailored to the specialised requirements of the application.

The following section explores the literature surrounding related work for SLV, piracy prevention and provenance.

2.7 Related Work

The previous sections in this chapter have discussed the scope of the software piracy issue, reviewed the software piracy process, and considered whether blockchain technology is a feasible method to provide user-based SLV.

This section explores literature for related works that propose methods and frameworks to prevent software piracy, and compares their functions and capabilities in order to help determine requirements for a SLV method. The outputs of this review will feed into both the Methods and the Requirements Engineering chapters.

The identified methods and frameworks for software provenance and software piracy prevention are outlined as follows. These present a range of approaches for preventing piracy in different use-cases.

Pirax: Khan et al. (2014) describes a framework for mobile App piracy using a cloud-based license server. When an App is first run on a mobile device, the App needs to register with the cloud license server to receive a unique license. This is achieved by creating a "node locked licensing scheme". The scheme utilises the phone's unique International Mobile Equipment Identity (IMEA) number, and the cloud server Universally Unique Identifier, to generate a unique registration serial number and an activation serial number for the App license. Each time the App is run, it verifies its encrypted activation serial number with the cloud license server, and if validation is successful, the App is allowed to run. It allows Apps to be authorised, or revoked, at the cloud license server. Khan et al. (2014) states that the Pirax framework mitigates several types of hacking, including preventing the App executing on a cloned image of the phone, and defeating a hacker extracting the App and executing it on another compatible platform. The Pirax high level architecture is shown in Figure 2.14.

DRM Framework: Veerubhotla and Saxena (2011) describes a theoretical common framework to protect digital objects, software libraries, and controlled software execution, with support for both online and offline models. The framework outlines 3 actors: (i) the publisher that produces a digital artefact to be protected; (ii) the consumer with an encrypted artefact downloaded from the publisher; and (iii) a DRM server that encrypts the publisher's artefact (and returns it to the publisher), and validates user license requests as shown in Figure 2.15 on page 87. The publisher needs to create a library for the core functions of the DRM functionality, and distributes it with the software. This becomes a bootstrap for the encrypted software, as well as being available after the software is decrypted. The confidentiality of artefacts and transport utilise a combination of X.509 certificates, and asymmetric and symmetric cryptographic keys, to support online and off-line capability. Veerubhotla and Saxena (2011) states several benefits for prevention of piracy, including secure communications, tamper-proof licenses, protection of cryptographic keys, and protection of software libraries and applications. Several non-functional benefits are also stated. However, these are primarily generalised assumptions reflecting architecture, and are not supported in the paper.

Tagged Transaction Protocol: This is a software license provenance model, created and defined by Palmer (2014), that enables anonymous transition of licenses across typical supply chain actors, such as suppliers, resellers, and customers. The provenance model defines a set of terms and relationships to record provenance of items in reseller chains, where actors in the supply chain are granted control at each stage of the supply chain that the item moves through. As each item moves through the supply chain, it is tagged; hence the model is called Tagged Transaction Protocol (TTP). The Tag Generation Centre (TGC) is responsible for creating

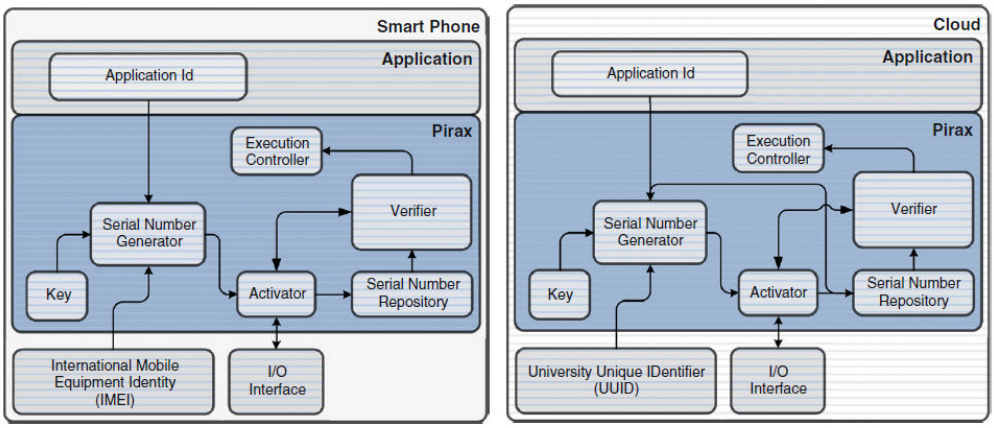


Figure 2.14: Pirax HLA
(Khan et al., 2014)

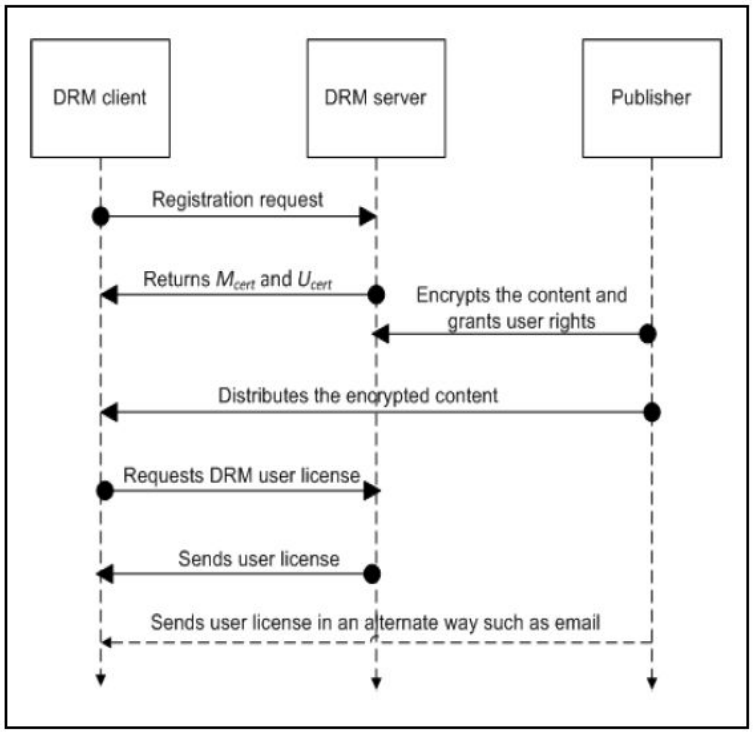


Figure 2.15: DRM Framework Sequence Diagram
Veerubhotla and Saxena (2011)

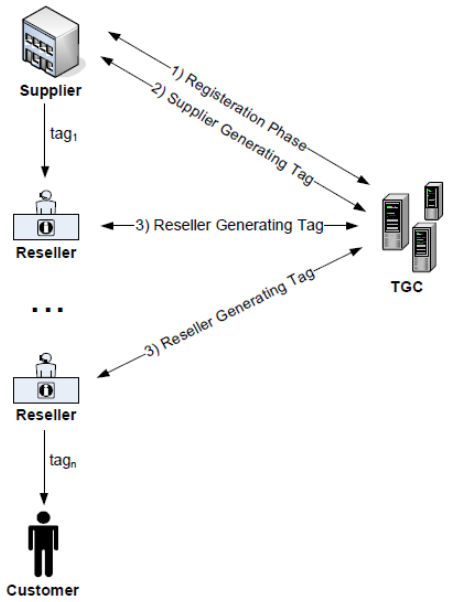


Figure 2.16: Provenance Graph for Tagged Transaction Protocol
Palmer (2014)

tags and securely transmitting them to each of the actors in the supply chain, as shown in Figure 2.16. Palmer (2014) states several benefits of TTP including prevention of spoofing, fabrication, cloning, identity revelation, linkability and protection from network sniffing.

Bitcoin: Munter (2016) remarks that Bitcoin may well be the "next big thing in the fight against piracy", citing industry interest and possible progress in utilising the decentralised nature of the blockchain to protect copyright. A review of literature reveals that a method has been proposed by Fortin (2011) called the Master Bitcoin Model (MBM), which demonstrates provenance using cryptocurrencies. A proof-of-concept was subsequently created by Lebo (2014), who named it "Dissent", and uses the Namecoin cryptocurrency (Ford, 2014) to demonstrate license provenance. This model uses the immutable nature of the cryptocurrency blockchain and the cryptographic proofs to validate that a user owns a specific

bitcoin that is linked to a license. It does not, however, provide any license validation in itself; it simply provides proof of ownership of a physical or digital asset.

Comparison and Discussion

Comparing the proposed methods above, most methods have functions such as: (i) generating a unique license for the client actor; (ii) utilising local device unique identifiers to prevent the App executing on unauthorised devices; and (iii) using public/private key encryption to protect the license, to validate the user, and to prevent the license from being copied. All methods use the Internet for ubiquity.

However, each method has key differences that limit its effectiveness as a SLV method. Pirax targets the mobile computing device platforms, has "node-locked licensing", and does not require any encryption of the App. The DRM Framework claims to provide piracy prevention for cross-platform and multiple content formats, but requires publishers to encrypt the application or content, and stores decryption keys on the server. TPP focusses on supply chain license provenance, providing license distribution and actor anonymity, but by nature is a one way method and does not protect from piracy once the client actor receives the license. The proposed Bitcoin method is fully distributed and provides anonymity, but requires a bootstrap application to access the blockchain, and only validates an entitlement to a license – it does not store any client actor or license metadata.

The first common element across all methods is the requirement for the user (or device) to prove that they are the actual entity, and validate themselves through a unique identifier, such as a private key or an embedded unique identifier. The second common element, is a license key that is bound to the validated user to establish entitlement. Together these form the entitlement for use of the software.

2.8 Problem Identification and Motivation

It is established that software piracy is significant problem from an industry perspective, with estimated US$132 billion losses per year. Also established is that the adversarial conflict between publishers and piracy actors continues, and piracy actors continue to develop exploits to break software piracy protections. Hence there is a strong motivation to address the piracy issue. To overcome the challenges presented by piracy actors, it has been determined that a ubiquitous SLV method is required, to validate user entitlements to the software at the time of use.

The objective of reviewing the literature on blockchain technology is to determine the feasibility of applying blockchain technology to create an SLV method. The findings of this literature review suggests that using blockchain technology as an approach to solve the piracy problem is both feasible and desirable. Blockchain technology, using the Internet as the medium for communication, provides the ubiquity to achieve the SLV objectives.

There already is a proof-of-concept in the MBM that shows that a cryptocurrency can be used to establish license provenance: a close requirement match for an SLV method. However, this presents some practical use-case limitations. MBM only conveys ownership; it does not have the ability to embed software licensing metadata or data such as license keys for specific entitlements, authorised end user information, and software feature and license upgrades. These limitations are caused by the cryptocurrency protocols and blockchain data structures. Therefore, to create an SLV method, defining a custom blockchain specifically to meet the requirements for SLV is required, and necessitates developing a NBA.

In summary, there is a strong motivation to determine if blockchain technology can be used for SLV. However, due to the scope and effort of creating an NBA, the objective of this research is limited to the design of the system and to establishing functional

requirements, rather than creation of a proof-of-concept. The system design provides a license entitlement through an NBA, overcoming the problems presented by the MBM, whilst meeting the non-functional requirements.[24]

This summary leads to the research questions and hypotheses to be addressed in this study.

2.8.1 Research Questions

RQ1: How can the Native Blockchain Application be applied, to meet the functional requirements of distributed SLV?

- **RQ1.1:** How can the Native Blockchain Application be utilised to protect software copyright by validating user entitlements?
- **RQ1.2:** How can the Native Blockchain Application validate user entitlements across multiple platforms?
- **RQ1.3:** How can the Native Blockchain Application be used to identify maliciously modified software?

RQ2: Which cryptocurrency provides the most suitable base for the SLV method?

- **RQ2.1:** What cryptocurrency characteristics are required to achieve data confidentiality in a blockchain-based SLV method?
- **RQ2.2:** What cryptocurrency characteristics are required to achieve actor privacy in a blockchain-based SLV method?

[24] Blockchain applications have many non-functional requirements, such as scalability, robustness, data integrity, and security. These are, however, beyond the of scope of this research, and are mentioned for completeness.

2.8.2 Hypotheses

Two hypotheses are proposed:

H1: To provide an SLV method, a Native Blockchain Application can be applied.

H2: In respect to SLV, the Native Blockchain Application enables a high level of data confidentiality and user privacy.

2.9 Conclusion

This chapter explores the literature surrounding the issue of software piracy, and whether blockchain technology can feasibly be utilised to overcome the challenges of protection of software copyright.

Through the review of literature on software piracy, it is shown that the scope of protecting software copyright from piracy is considerable. The attack surface for piracy includes physical media, digital media, diverse hardware platforms, multiple operating systems and even cloud services. Protection methods for software are routinely circumvented by technically savvy crackers, whilst software end user license agreements are consistently breached, both deliberately and accidentally. The outcome is that software piracy remains rampant, with an estimated economic cost of US$132 billion per year. The outcome of the research is that a method is required to validate user entitlements to software, regardless of platform used or location of the user.

Blockchain technologies are reviewed and have been identified as a feasible technology to overcome the piracy issues. Applications can be written to utilise public cryptocurrencies such as BitCoin and Ethereum, or can be designed from the ground up as a new application with its own dedicated blockchain (NBAs). Protecting software copyright utilising an NBA to overcome the software piracy challenges is feasible, as existing proof-of-concepts and industry developments demonstrate new use-cases for

blockchain technology. This research presents an interesting opportunity to explore the functional requirements of SLV utilising blockchain technology.

In the next chapter, the identification of the appropriate Research Methodology is discussed. This includes an overview of potential research methods and a discussion of the scope of the cryptocurrency and blockchain domain in respect to research. Functional and non-functional requirements are briefly explained, for the purpose of supporting the research methodology selected to test the hypotheses.

Chapter 3

Method

3.1 Introduction

In the previous chapter the problems surrounding software piracy were discussed, and the scope of the challenges showed that a user-based SLV method was required to address the problem. Cryptocurrency and blockchain technologies were then examined to establish the feasibility of using blockchain technology for SLV. Examples of NBA use-cases by industry demonstrate that NBAs are used to provide ubiquitous, and secure, access to data over the Internet, and may be applied to for SLV.

In this chapter, the appropriate Research Methodology to test the hypotheses and research questions, is determined. This commences with brief investigation of blockchain application use-cases across different technology domains, to identify research methodologies from a similarly aligned domain. RE and FD software engineering requirements are considered in the context that the output of this study is to design a system. Formal Methods and Experimental Design Research (EDR) are discussed. This then leads into a discussion on DSR, and its suitability to the research outcomes of this study, and supporting the research methodology, to test the hypotheses.

3.2 System Design Considerations

To test the hypotheses, design of the system will necessitate a software engineering process. This will include RE to describe the blockchain license validation requirements, and FD to describe the function specifics for SLV. To determine the appropriate research methodology for this study, a review of the literature is conducted. The focus of the review is to identify software engineering research methodologies used for systems and architectures similar to NBAs.

The software engineering requirements for development of distributed ecosystems are likely to differ from those of centralised software (Shehory & Sturm, 2014). This is apparent as in principle, centralised software executes within a single operating system environment and utilises the resources within that execution environment. On the other hand, distributed systems are factored, necessitating further functional and non-functional requirements. These include requirements such as: protocols for inter-node communication, defining interaction mechanisms, architecture of the distributed system, frameworks, and standards, as shown in Figure 3.4 on page 111. Cryptocurrencies and NBAs are multiple actor ecosystems (Alqassem & Svetinovic, 2015) and, as such, utilise several of these elements from Agent Oriented Systems Engineering.

For the purpose of determining the most appropriate research method for this research, it should firstly be recognised that cryptocurrencies and blockchain technologies are multiple actor ecosystems (Alqassem & Svetinovic, 2015), with research across multiple domains intersecting with cryptocurrencies and blockchain technologies: research other than software engineering. The nature of these intersecting domains will influence research methodology selection. Hence this research will need to consider what domain is sufficiently aligned with the research objectives. Domains that intersect with cryptocurrencies and blockchain technologies include:

1. Artificial Intelligence: where the blockchain can be used to aggregate data, and

agents process the data and act accordingly (AI Blockchain, 2017; Noyen et al., 2014).

2. Banking: where both banking industry and governments are considering the application of blockchain for tax collection and eliminating fiat currency (Hochstein, 2014; European Central Bank, 2015; Keith et al., 2016; World Economic Forum, 2015).
3. Crime and fraud: criminals and malicious actors have been using cryptocurrencies for anonymous payments and evasion of authorities (Bryans, 2014; EUROPOL, 2014; Kirkpatrick, 2017).
4. Cryptography: this is the core of cryptocurrencies and blockchains, especially around proofs and maintaining integrity of the blockchain (Buterin, 2017; Percival, 2009; Eddy, 2016).
5. Data sciences: utilising data exploration and visualisation technologies on the blockchain and providing analyses (Maesa, Marino & Ricci, 2016; Molina-Solana, Birch & Guo, 2017).
6. Distributed computing: utilising the computing power of the cryptocurrency ecosystem for large computational projects (Halford, 2014; University of Sheffield, 2016).
7. Economics: similar to banking, but in an macro-economic and reserve banking context (European Central Bank, 2015; Danezis & Meiklejohn, 2015).
8. Finance: where financial institutions leverage the smart contract capability of cryptocurrencies for trading and digital representation of tangible assets (Keith et al., 2016; Bashir, Strickland & Bohr, 2016; Underwood, 2016).
9. Identity: where user identity, access, and authorisation, is managed through the blockchain (Shea, 2017; Underwood, 2016; Mainelli, 2018).
10. Law: governments are challenged to align virtual currencies and peer-to-peer contract law, both within local statute and across multiple jurisdictions, and

need to adopt and/or regulate cryptocurrencies (Marian, 2014; Financial Crimes Enforcement Network, 2014).

11. Mathematics: where mathematical and formal methods are used to logically describe and validate cryptocurrency and blockchain functions (Sprankel, 2013).
12. Security: there are many aspects of security across the actors of cryptocurrency and blockchain ecosystems (Courtois & Bahack, 2014; Porter, 2015; Dunn, 2013).
13. Social impact: similar to social media, cryptocurrencies and blockchain technology have the capability for significant social impact (Chakravorty & Rong, 2017; Bashir et al., 2016; Bill and Melinda Gates Foundation, 2014).
14. Sustainability (in two contexts): eco-friendly benefits by reducing the carbon footprint and the long term viability of the cryptocurrency as it grows (McCook, 2014; Chuen, 2015).

In considering the above domains, the closest alignment for this research is Identity, where the user identity is linked to license entitlements and associated data, cryptographically stored in the blockchain. For the purposes of clarity in this research, the blockchain-based SLV method being researched will be referred to as "ReSOLV".

3.3 Overview of Requirements

Although chapter 4 discusses the RE process for ReSOLV, consideration of typical requirements should be made as these may influence the determination of the most suitable research method.

In the first instance, chapter 4 identifies that SLV methods and frameworks have two layers that relate to functional requirements. These functional requirements are derived from the methods and frameworks identified in Section 2.7 on page 85, Related Work. The two layers identified are: (i) the human actor layer that represents the

users, merchants and vendor/publishers; and (ii) the technology actors such as servers, databases and protocols that provide the SLV functions. Secondly, chapter 4 identifies many non-functional requirements that cryptocurrencies and blockchains share, and also identifies requirements specifically for a ReSOLV method as discussed in Section 2.7. These include: (i) Design requirements (scalability, robustness, resiliency, data integrity and security); (ii) Usability requirements (independence, mobility, portability and flexibility, and standalone); and (iii) Governance considerations (hostile takeover, ownership, stability, economic design, and code source).

In comparison, a ReSOLV method is likely to require the same human actors as the SLV method, as the objective to prevent piracy is the same for ReSOLV. However, from a technology actor perspective, server and database are replaced with miner and blockchain respectively. Hence, a server-centric method is replaced with a distributed method [1]. SLV non-functional requirements will be similar to ReSOLV requirements. However, determining these requirements in a distributed cryptocurrency/blockchain ecosystem requires investigation to select appropriate implementation and approach for the non-functional requirements in the ReSOLV ecosystem.

Hence, selection of the most suitable research methodology must consider similar requirements for SLV, but applied to a ReSOLV blockchain method for a successful outcome. The following section assesses some selected research methodologies for suitability, given the functional and non-functional requirements.

[1] The distributed entities that provide the blockchain functions as shown in Figure 2.9 on page 72.

3.4 Methods for Consideration

There are now a number of uniquely developed cryptocurrencies [2] and NBAs. Examples these include NXT[3], Ethereum[4], BlackCoin[5], Maidsafe[6] and StorJ[7]. However, from a research perspective, for most of these examples the requirements definition for the cryptocurrency or NBA is established, at best, through self-published white papers, which primarily serve as a high-level design discussion instrument. There is a single self-published Ethereum cryptocurrency research paper, providing a complete formal methods definition of Ethereum (Wood, 2016), yet providing no illumination as to any initial research or software engineering methodologies.

It seems that, as these cryptocurrencies and NBAs are industry initiatives, research methodologies do not have much weight. The high-level white papers in the unique cryptocurrency and blockchain applications intimate that the software engineering process takes a highly agile design approach, whilst the only research paper takes a formal methods approach, well after the software engineering is complete.

The following sections consider options for suitable research methodologies to test the hypotheses and research questions. Formal Methods and Experimental Research are discussed, but do not suit the proposed research objectives, due to the iterative design and engineering requirements. Subsequent sections then discuss (i) DSR, as the preferred research methodology; (ii) Agile methodology and Behaviour Driven Development, as the preferred approach for software engineering; and (iii) and Agent Oriented Software Engineering, as a potential framework for designing a NBA distributed ecosystem.

[2] Cryptocurrencies other than Bitcoin designed and developed from the ground up, rather than a cryptocurrency that uses an existing open source base cryptocurrency with a few modifications.
[3] NXT https://nxt.org/
[4] Ethereum https://www.ethereum.org/
[5] BlackCoin http://blackcoin.co/
[6] MaidSafe https://maidsafe.net/
[7] StorJ https://storj.io/

3.4.1 Formal Methods

Formal methods require a mathematically rigorous semantics basis to define software (FormSERA, 2012). Not commonly used in software development (Gnesi & Plat, 2015), formal methods, however, have many justifications for use in situations that require a higher quality software output (Hinchey et al., 2008). Examples include use in RE (Laplante, 2014), critical systems development (Wolff, 2012), addressing scalability problems using statistical model checking of concurrent systems (Agha, 2013), and software product line engineering (Schaefer & Hähnle, 2011).

Supporting this, Laplante (2014) observes that development using informal techniques and natural language can result in misinterpretation of an element, requirement, or specification, of a system. Even a single punctuation error or missing character can cause a significant problem. In 1962 a missing hyphen character in a FORTRAN code statement led to the loss of the Mariner 1 spacecraft, the first American probe to Venus (Laplante, 2014). More recently, in 2017, new cryptocurrency Zcoin[8] lost $585,000 due to one extra character left in the open source Zerocoin source code (Kumar, 2017).

Formal methods are applicable in where the outcomes require reliability and robustness. However, most of the examples of formal methods in software engineering pertain to either improving existing systems, or carefully defining the requirements and testing outcomes (Hinchey et al., 2008). Integration of formal methods into Agile software engineering, whilst promising, needs to be explored (Wolff, 2012). In the case of Ethereum, Formal Methods have been used to describe the entire Ethereum function once it was operational, rather than as a software engineering methodology at the commencement (Wood, 2016).

Formal methods shows a strong use-case, as cryptocurrencies and NBAs mature, and non-functional requirements such as reliability, scalability and security become

[8] Zcoin http://zcoin.tech/

essential. However, formal methods requires a significant amount of effort to detail the design and outputs during the software engineering process (Hinchey et al., 2008). From an NBA perspective, designing a system with no existing artefacts is likely to require a highly iterative approach to get the functional basics completed. For this reason, it is concluded that formal methods is not a suitable research methodology to achieve the research outcomes.

3.4.2 Experimental Design Research

Experimental Design Research is considered because it is used in Information Systems (IS) research. Experimental design is based on observation (Thurimella, 2014) of existing artefacts and systems. Levy, Levy and Ellis (2011) states that experimental design includes four research categories. These include: lab experiment, quasi-experiment, factorial design and the ex-post facto design. These are primarily focussed on involving grouped participants and a control group, and require functioning artefacts or systems to undertake the research.

Hence, given that Experimental Design Research requires an artefact to work with in the first place, it is not suitable for the type of artefact creation research that the objectives of this research require.

3.4.3 Design Science Research

DSR is an IS research methodology that is generally accepted as a legitimate approach to IS research. As IS developed in the 1980s, a need for systems development research was established in light of IS systems being composed of social-technical artefacts involving people and technology systems (Silver, Markus & Beath, 1995). Nunamaker Jr. and Chen (1990) proposed a new framework of research for computing and computer application research, presenting a Process for Systems Development Research that

parallels research methodologies used by social and behavioural sciences. This five-step research process requires: (i) constructing a conceptual framework; (ii) developing a system architecture; (iii) analysing and designing the system; (iv) building the prototype system; and (v) observing and evaluating the system (Figure 3.1).

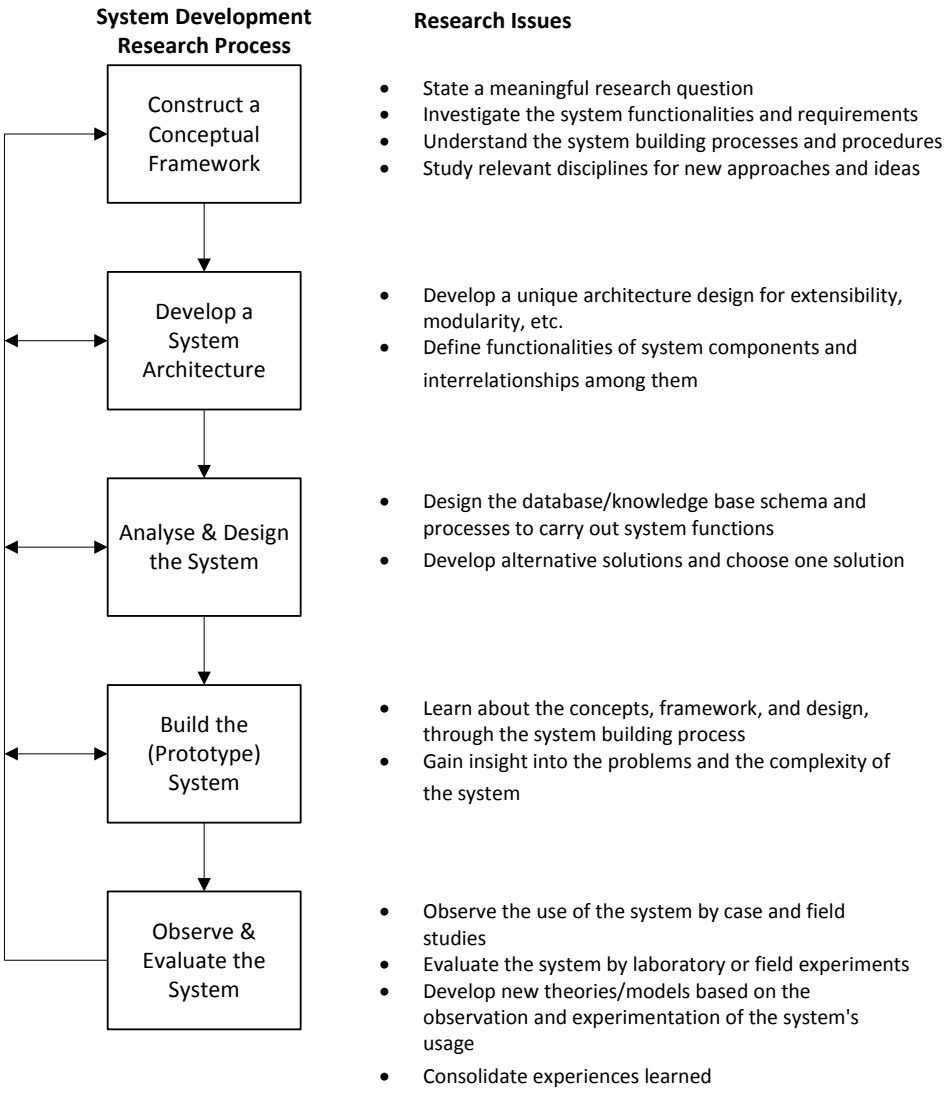


Figure 3.1: System Development Research Process (Nunamaker Jr. & Chen, 1990)

Literature has progressively built the case for DSR as a valid research methodology for IS and software development (Hevner, March, Park & Ram, 2004; Peffers, Tuunanen,

Rothenberger & Chatterjee, 2007; Gregor & Hevner, 2013). In DSR, researchers engage with developers to design and produce an artefact that addresses a specific problem (Papas, O'Keefe & Seltsikas, 2012; Gregor & Hevner, 2013) or, to attempt to construct new and innovative artefacts or solution technologies (Iivari & Venable, 2009). Hevner et al. (2004) states that "effective DSR should make clear contributions to the real-world application environment from which the research problem or opportunity is drawn". Hence, DSR is actively employed for Agile development projects, which take an iterative approach to create artefacts based on high-level design or user requirements (Kautz, 2011; Brhel, Meth, Maedche & Werder, 2015).

	Contribution Types	Example Artifacts
More abstract, complete, and mature knowledge	Level 3. Well-developed design theory about embedded phenomena	Design theories (mid-range and grand theories)
↕ ↕ ↕ ↕	Level 2. Nascent design theory—knowledge as operational principles/architecture	Constructs, methods, models, design principles, technological rules.
More specific, limited, and less mature knowledge	Level 1. Situated implementation of artifact	Instantiations (software products or implemented processes)

Figure 3.2: DSR Contribution Types (Gregor & Hevner, 2013)

To determine whether a DSR research objective is contributing sufficiently to a body of knowledge, Gregor and Hevner (2013) posits a DSR knowledge contribution framework, as shown in Figure 3.3 on page 105. This framework is built on understanding DSR contribution types as shown in Figure 3.2, where abstraction of knowledge in DSR is considered. This shows research contributions can be undertaken for "more specific, limited and less mature knowledge" research (Gregor & Hevner, 2013), through to "more abstract, complete and mature knowledge" research (Gregor & Hevner, 2013). Three levels of contribution types are defined, providing guidance for researchers to evaluate the type of contribution that their research, and to determine the type of artefact output expected for the research.

Gregor and Hevner (2013) states that the framework for knowledge contribution of a design research project is predicated on the research skills of the team in appropriately drawing "useful knowledge" from both descriptive and prescriptive knowledge bases. Descriptive knowledge includes: (i) observing, classifying, measuring, and cataloguing phenomena; and (ii) making sense of natural laws, regularities, principles, patterns, and theories. Prescriptive knowledge includes constructs, models, methods, instantiations, and design theory. Gregor and Hevner (2013) also points out that a fundamental issue in DSR is that nothing is really new. Everything is made out of something else, or builds on some previous idea, so how can a researcher determine when is something really novel or a significant advance on prior work?

With consideration of Figure 3.2 on the preceding page, a dimension of maturity in respect to the research is recognised by Gregor and Hevner (2013) in Figure 3.3 on the next page. This reflects the research project's placement along the timeline of knowledge growth in the discipline, and is related to the problem maturity and solution maturity available and relevant to the DSR project. The quadrant model in Figure 3.3 on the following page outlines the types of contribution that can be made in respect to different levels of DSR. These are described as follows:

- **Routine Design:** reflects application of known solutions to known problems and, as such, does not have a major research knowledge contribution.
- **Improvement:** reflects the development of new solutions for known problems, resulting in a research opportunity and knowledge contribution.
- **Exaptation:** reflects extending known solutions (e.g. from other fields) to new problems, and is considered a research opportunity and knowledge contribution.
- **Invention:** reflects a new solution for a new problem, and hence also a research opportunity and knowledge contribution.

Hence, the DSR Knowledge Contribution Framework provides the rationale for selecting

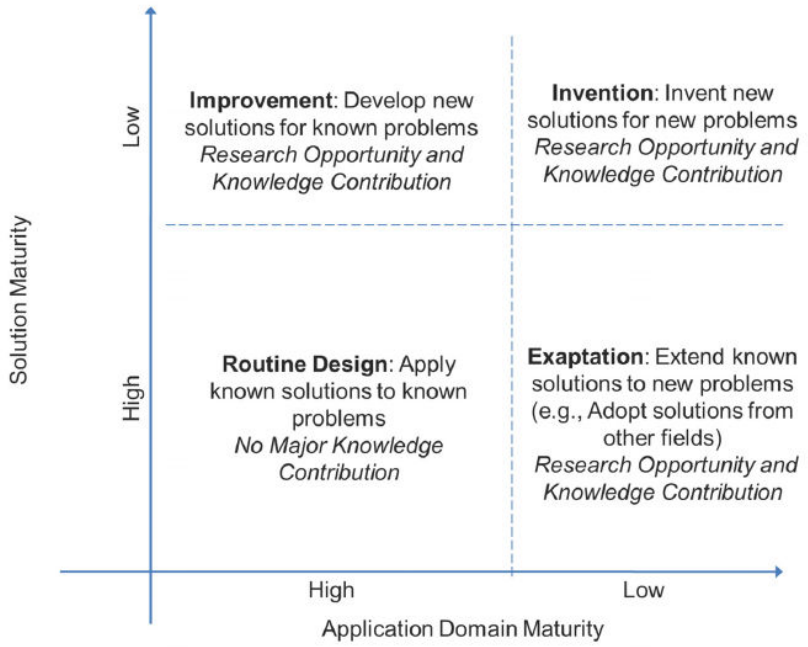


Figure 3.3: DSR Knowledge Contribution Framework (Gregor & Hevner, 2013)

DSR methodology as the most suitable research methodology to address research objectives. The remaining requirement is to determine the appropriate quadrant for the research objectives, and provide a compelling argument for this selection.

Blockchain technology is a nascent domain being recognised by industry as a disruptive technology. There has been a relatively small amount of academic research opportunity, due to newness of the domain and rapidity of evolution of blockchain technology. Application of blockchain technology to solve existing problems is resulting in both innovative and novel outcomes. The contribution type of the proposed research can be classified as level 2 and level 3 according to Figure 3.2 on page 103, reflecting the nascent design theory with outputs including constructs, methods, models, and design principles. The research objectives are thereby classified in the Invention quadrant of the DSR Knowledge Contribution Framework, and are considered a valid DSR opportunity.

3.4.4 Summary

This section reviewed the methodologies considered feasible to test the hypotheses and research questions. DSR has been established as the most appropriate methodology for this research and has met the knowledge contribution requirements, as well as the innovation and novel characteristics, to be classified as an Invention.

The following section discusses DSR software engineering using Agile methodologies, with a focus on behaviour driven development as the preferred approach.

3.5 Design Science Research Discussion

In the previous section, formal methods, experimental methods, and DSR were considered, to determine the most appropriate research methodology to test the hypotheses and research questions. The DSR approach to defining knowledge contribution and solution/application domain maturity demonstrated that DSR was the most suitable methodology for this research.

This section discusses Agile software engineering as the most appropriate software engineering approach for blockchain artefact creation. It discusses Behavioural Driven Development as the preferred RE methodology, explaining the use-case for "as-a-user" and "given-when-then" (GWT) user stories. Agent Oriented Software Engineering (AOSE) is also discussed as a software design methodology for distributed systems, drawing parallels with the blockchain ecosystem.

3.5.1 Agile Software Engineering

Cryptocurrency and blockchain research and development has been undertaken primarily by industry groups, such as the Bitcoin Foundation and Ethereum, using an Agile software development approach. Commercial and finance businesses are also actively

developing cryptocurrency and blockchain applications[9] [10]. Furthermore, Lewis (2016) argues that only an agile/lean approach for blockchain development is viable, as a rapid release cycle and immediate validation is the only way to approach a challenge such has blockchain development.

The nature of the distributed peer-to-peer cryptocurrency ecosystem means that there are multiple digital entities and protocols that need to be designed, developed, and tested, both individually, and within the ecosystem. Unfortunately, due to the novelty of cryptocurrency and blockchain, access to experienced development resources is very limited, and open source code often lags the latest cryptocurrency releases. Furthermore, most source code does not have architecture or design documentation, creating a reliance on reading the source code and understanding design principles embedded within.

As a result of the combination of the novelty and the complexity of the ecosystem, the development methodology for this research will need to be highly agile and iterative. The software engineering process, of necessity, needs to support the design of individual artefacts that can be tested as part of the developing ecosystem. Given the challenges and requirements, and industry software engineering precedent, Agile development methodologies are clearly the appropriate software engineering approach to undertake this research.

3.5.2 Behaviour Driven Development

The purpose of this section is to establish the appropriate Agile RE documentation process for the ReSOLV blockchain prototype.

User stories, prototypes, use-cases, scenarios, and story cards are key artefacts for the documentation of requirements that are used in RE (Schön, Thomaschewski &

[9] Applied Blockchain https://appliedblockchain.com
[10] IBM https://www.ibm.com/blockchain/offerings.html

Escalona, 2016). Based on many case studies Schön et al. (2016) finds that user stories, prototypes and use-case scenarios are the most frequently used artefacts in Agile RE, with user stories clearly being the preferred RE documentation practice. In discussing approaches to user stories, Dimitrijevic, Jovanovic and Devedzic (2015) finds that there is no unique user story process, with different RE practices proposed by several authors. The authors also state that user stories are not considered to be artefacts of analysis activities, but rather the medium for communication between customer and developers. Each user story must articulate the requirement clearly and succinctly, eliminating ambiguity and supporting a test case.

Hence, a well constructed user story needs to fulfil multiple objectives. It should have enough details to understand the business intent, the actor roles, the goal, and the depth, so that the team can understand it to produce a workable prototype or software, and the business can understand it (Adzic, 2015). User stories generally follow the INVEST model: (1) Independent, (2) Negotiable, (3) Valuable to users and customers, (4) Estimable, (5) Small, and (6) Testable (Dimitrijevic et al., 2015).

Most user story templates are focussed on identifying user RE between customer and developer: for example, the common "as-a-user, I want" story template. This template does not fit well with service or agent based systems such as blockchain ecosystems, as these systems are not heavily user oriented. Rather, they are behaviour oriented, where the system is event driven, rather than user driven. Behaviour driven story templates, such as GWT, provide for pre- and post-conditions that trigger a user story event, and allow for easy testing and acceptance criteria (Adzic, 2015).

Specifications should be self-explanatory. GWT is a structured format for clearly expressing scenarios with example data, including pre- and post-conditions. This provides the capability to express RE, based on events that may trigger within the blockchain ecosystem. Adzic (2015) states that when used correctly, GWT helps teams design specifications and checks that are easy to understand and maintain. As tests will

be focused on one particular action, they will be less brittle and easier to diagnose and troubleshoot. When the parameters and expectations are clearly separated, it's easier to evaluate when more examples are needed, and to discover missing cases.

To illustrate the differences between user and behaviour driven user stories, two examples are provided for the same user story (below): (1) "As-a-User", and (2) "Given-When-Then".

Objective

The objective in this example is for a ReSOLV Published App (PApp) to execute the software, after checking if the user has already successfully authenticated to the ReSOLV User Wallet (UW), and that there is a valid software license residing in the UW.

"As-a-User" User Story Type

- As a user, I want the PApp to check my ReSOLV UW, to validate that I have already successfully authenticated and that there is a license for the software, so that the PApp will execute the software.

"Given-When-Then" User Story Type

- GIVEN that I have already successfully authenticated to my UW
- WHEN there is a valid license for the software in my UW
- THEN the software will execute.

Comparing the two user story types, it is shown that the GWT method offers a very succinct condition-oriented structure and clarity for unit testing, whereas the "as-a-user" method results in a description of higher complexity, requiring more effort to create a separate unit test.

In summary, Behaviour Driven Development is clearly the more suitable RE methodology for this research.

3.5.3 Agent-Oriented Software Engineering

The investigation into a suitable software design methodology for distributed systems identified the field of AOSE as having distinct parallels to the blockchain ecosystem. Wooldridge and Jennings (1995) introduces the concept of agents providing intelligence and knowledge, and proposes architectures and frameworks. Shehory and Sturm (2014) explores the field of AOSE, as it has evolved over two decades since its inception, and discusses the core AOSE themes, and distinctions between agents and centralised software. Agents and multiple agent systems (MAS) are used where centralised software does not provide sufficient flexibility or capability to meet requirements, and functionality is separated accordingly, often dictated by the location of the software.

Shehory and Sturm (2014) establish that agent-based systems focus on dynamically integrating components. They have their own threads and controls, are engaged in complex protocols, and have an emphasis on interaction of the components. Five primary themes have been evolved from these concepts, in the form of methodologies, modelling techniques, framework implementations, agent-programming languages and agent communication. These themes are shown in the thematic map in Figure 3.4 on the following page.

In addition to the five themes, Shehory and Sturm (2014) outlines the four dimensions of agents, which are: (i) autonomy; (ii) intelligence; (iii) sociality; and (iv) mobility. These are described below.

- Autonomy refers to the "ability of an agent to perform unsupervised computation and action, and to pursue its goals without being explicitly programmed or instructed"
- Agent intelligence includes capabilities such as learning, reasoning, planning and decision-making. These, in turn, allow the agent to have abstract goals and to act accordingly on internal and external data, events and information. These are

Figure 3.4: Agent-Oriented Software Engineering Themes (Shehory & Sturm, 2014)

known as Belief, Desire, Intention (BDI) agents (Wooldridge & Jennings, 1995).

- Sociality reflects the multi-agent environment, where agents may need to interact with each other to co-ordinate, collaborate or compete.
- Some multi-agent systems exhibit mobility. Agents may change their logical or physical location (move from one execution environment to another execution environment), or the system the agent resides on is mobile, such as a smartphone.

In summary, software that executes in the cryptocurrency and blockchain ecosystem exhibits all four dimensions of MAS, and therefore research methodologies such as AOSE should be strongly considered for blockchain ecosystem research. However, due to the complexities of designing an AOSE or blockchain ecosystem, the AOSE software design methodology is not within the scope of this research.

3.6 Conclusion

The proposed research methods identified for each of the research questions, linked to the hypotheses, are shown in Figure 3.5. Design Science is the research methodology selected to determine the validity of the hypotheses, and will be used to demonstrate how each of the research questions can be confirmed using cryptocurrency or blockchain systems.

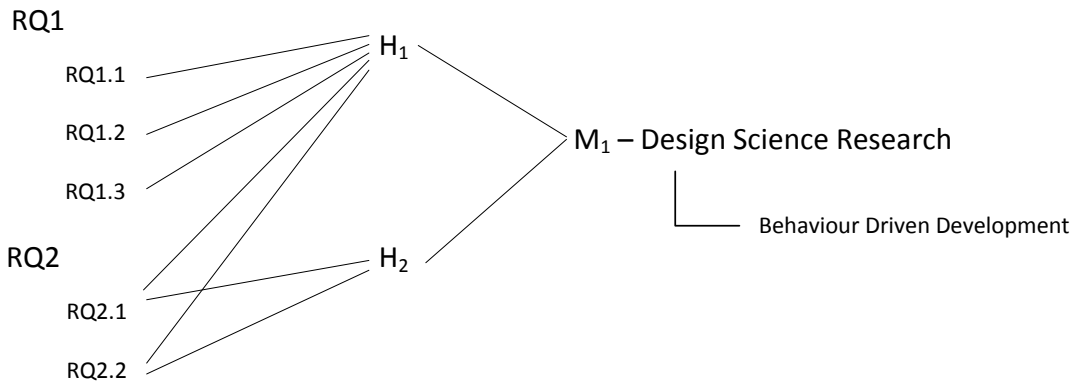


Figure 3.5: Study Research Methodology

Chapter 4

Requirements Engineering

4.1 Introduction

In the previous chapter, suitable research methodologies for evaluating cryptocurrency and blockchain related research were explored. Three methodologies were selected for consideration: Formal Methods, Experimental Design Research, and DSR. Further consideration was given to Agile RE methodologies and similar SLV constructs, to provide more research related insights. The outcome of this evaluation resulted in DSR being determined to be the most suitable research methodology to test the hypotheses and research questions.

In this chapter, the approach to RE for ReSOLV is discussed, and a requirements specification developed. Definitions and the general RE process are established and outline the activities that are involved in the RE process. The ReSOLV model is presented, building on the previously identified MBM, demonstrating the main constructs for SLV on the blockchain. The requirements specification establishes functional and non-functional requirements for the ReSOLV model, culminating in the design of the ReSOLV HLA and RA, which describes the core functionality of the ReSOLV method.

Finally, user stories for the ReSOLV RA are constructed using a Behaviour Driven

Development approach. The output of this RE section are artefacts for the following:

- HLA for the ReSOLV ecosystem
- RA for the core ReSOLV processes
- Requirements specification (user stories) for the RA

4.2 Definitions and Process

Hull et al. (2011) asserts that software is the dominant force of change of new products. Systems are becoming more complex and have deeply integrated software components, whilst the demand for instant distribution pressures the development and deployment methodologies. The challenge is for developers to create the "right product" within the constraints set out by business, or the requirements outlined by the client. RE is the discipline that is designed to improve quality and ensure successful outcomes are achieved for stakeholders. For discussion of RE, it is important to distinguish between requirements and goals, and to ensure that these concepts are identified and separated for forward clarity. Laplante (2014) states that goals are high-level objectives of a business, organisation, or system. However, a requirement specifies how a goal should be accomplished by a proposed system. Furthermore, Sommerville (2010) observes that the term 'requirement' is not used consistently in the software industry. This is often due to the size and nature of the project, requiring a high level of abstraction at a business level, which necessitates further analysis to identify complex interactions, processes and workflows.

The definition of RE has evolved since the mid-1990s, and some examples of these evolving definitions are provided below:

Zave (1997, p. 315) provides a definition for software RE, described as "more-or-less

universal for RE" by Laplante (2014): "Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software families."

IEEE Computer Society (IEEE, 1998) defines RE as: "the process of defining, documenting, and maintaining requirements for a program. This can be applied to systems and software engineering".

BSI Standards (2011) defines RE as: "an interdisciplinary function that mediates between the domains of the acquirer and supplier to establish and maintain the requirements to be met by the system, software or service of interest."

Adzic (2011) states RE is defining scope and building the right scope: "Requirements engineering is concerned with discovering, eliciting, developing, analysing, determining verification methods, validating, communicating, documenting, and managing requirements."

The evolution of RE is reflective of the expanding role of software. As software becomes more complex and has more moving parts[1], identifying requirements is increasingly difficult as requirements become more abstracted out of necessity. RE is used to help deconstruct the high level abstraction business layer into manageable, understandable, and sufficiently detailed requirements across the various platforms and layers.

RE is separated into functional requirements, which include user requirements and system requirements, and non-functional requirements (Sommerville, 2010). Functional requirements are descriptions of services the system should provide, its inputs, outputs, and behaviour in particular situations. User requirements are described in a natural language form that can be understood by users (in Agile methodology these are user

[1] Examples include cloud computing, Internet of Things, Agents and Interfaces

stories), whereas system requirements describe function inputs, outputs, and constraints in detail. Non-functional requirements are not related to specific services, but may relate to system properties, such as reliability and response time, or characteristics of the system as a whole, such as performance or security (Sommerville, 2010). As non-functional requirements apply to the entire system, they may be considered more critical than individual functional requirement (Laplante, 2014). Figure 4.1 outlines types of non-functional requirements that may be essential to incorporate into the RE process at the outset.

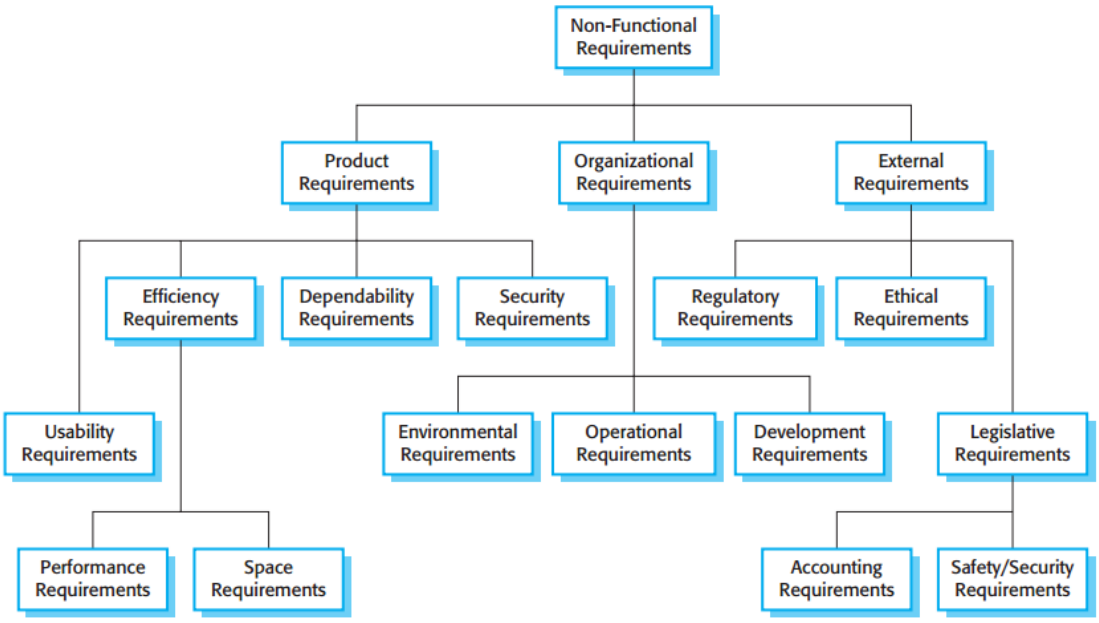


Figure 4.1: Types of Non-functional Requirements (Laplante, 2014)

Establishing requirements is a vital part of the software engineering process, and selecting the appropriate RE process is a critical step towards achieving a successful development outcome. Both functional and non-functional aspects should be considered in any blockchain development, due to the multiple actor interactions in the distributed ecosystem. The method for defining the RE of the blockchain validation artefact is

discussed in Section 4.4 on page 131. The RE process comprises a set of activities that aims to create consensus amongst the stakeholders, and establishing a requirements document that satisfies stakeholder requirements (Laplante, 2014). The activities involved in the RE process vary widely, depending on the type of system being developed and the specific practices of the stakeholders involved. The RE process consists of four main activities, as described below and shown in Figure 4.2 on the next page.

1. Feasibility Study: assessing that the proposed system is realistic within established business and financial constraints.

2. Requirements Elicitation and Analysis: checking functional requirements, resolving stakeholder conflicts, and systems modelling (e.g. Unified Modelling Language (UML) sequence diagrams).

3. Requirements Specification: creating user requirements (user stories) and systems requirements.

4. Requirements Validation: checking that the documented requirements and models are consistent and meet stakeholder needs.

Requirements elicitation and analysis has a wide range of methods of engagement, including face to face engagement with stakeholders such as customers, sponsors, suppliers and technical personnel. The purpose of using a variety of elicitation methods is to establish a holistic view of requirements from different stakeholders, based on their particular lens or perspective of the project. To identify the full gamut of requirements often necessitates different approaches to build a complete and accurate picture of needs of all the stakeholders and systems. However, for this research, the requirements elicitation is primarily based on constructing a ReSOLV model that derives its functions from earlier non-blockchain based SLV work. There are no interactions with stakeholders

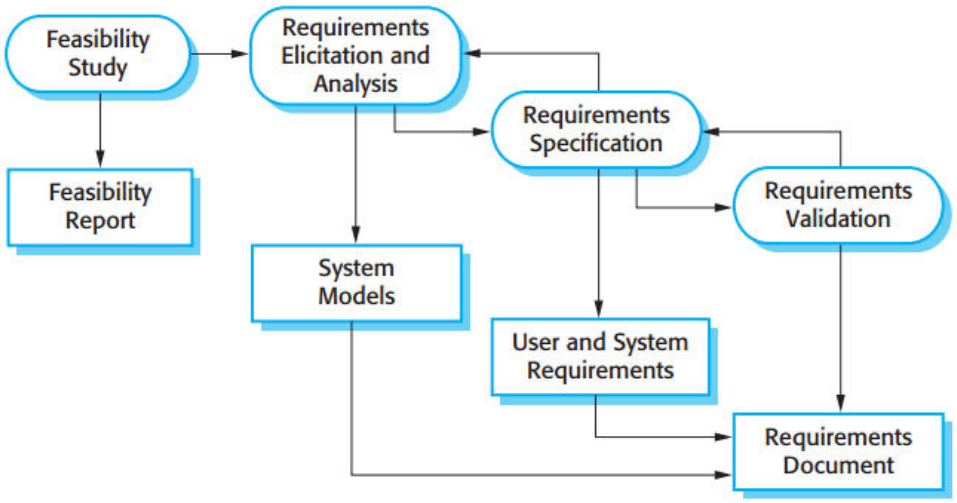


Figure 4.2: RE Process (Laplante, 2014)

such as end users or software vendors, as the guidance from the earlier, non-blockchain based SLV work sufficiently describes the needs of these actors.

The Requirements specification introduces Agile methodologies and uses the principles outlined in the Agile Manifesto[2]. These principles are based on creating more value in the software engineering process and uncovering better ways of developing software (Williams, 2012), as described below:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

To define the ReSOLV model, user stories will be constructed to describe the various functions and methods that will be required by the actors in the blockchain ecosystem. These functions will subsequently be defined in detail in Chapter 5, Functional Decomposition.

[2] http://agilemanifesto.org/principles.html

For the purpose of this research and due to time constraints, the activities that this research will focus on are stated as follows: (i) creation of artefacts for the Requirements Elicitation and Analysis (Model and Architecture); and (ii) the Requirements Specification.

The following section discusses Requirements Elicitation for SLV. It first explores the MBM (See Section 2.7 on page 85) license provenance functionality and discusses its application to ReSOLV. The ReSOLV Model and ReSOLV HLA for SLV are introduced. The ReSOLV Model discusses the fundamental operation of SLV using the blockchain, building on the MBM, whilst the HLA describes the architecture and vision for ReSOLV and outlines how the broader elements interact in the blockchain ecosystem.

4.3 Requirements Elicitation

4.3.1 Master Bitcoin Model

The MBM, proposed by Fortin (2011), is a basic form of SLV. Using Namecoin as the underlying blockchain, the model has been implemented by Lebo (2014) as a proof-of-concept project called “dissent”. In this model, the vendor address/bitcoin combination represents license ownership, and if the user has a transaction that shows the bitcoin originated from a specific vendor address, then the user is considered to have ownership of the software. This concept is demonstrated in Figure 4.3 on the next page.

The entities illustrated in Figure 4.3 are as follows:

- Vendor1 (`V1`): owns the Software application S1
- Software1 (`S1`): the particular Software application
- MasterAddress1 (`M1`): the address representing S1
- UserAddress1 (`U1`): the end user address for the wallet that holds the bitcoin indicating software entitlement.

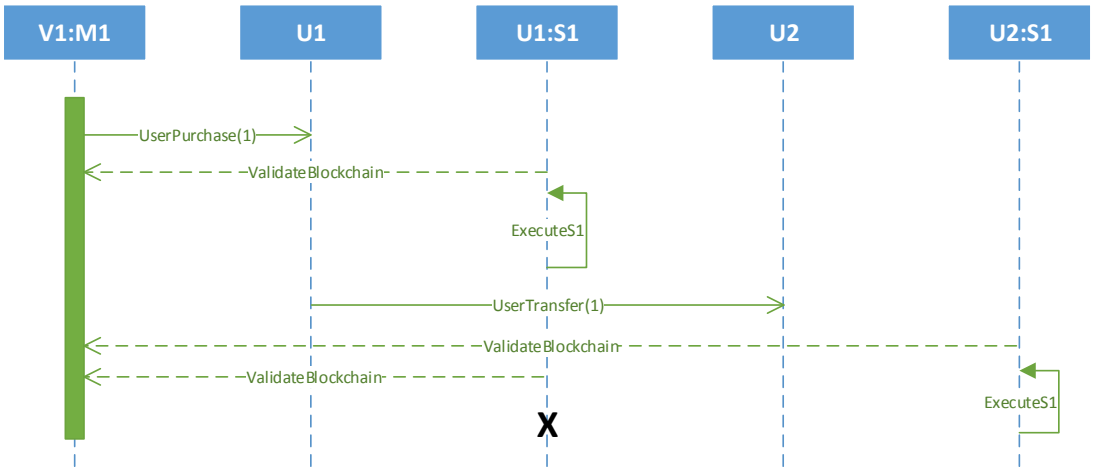


Figure 4.3: MBM Transfer of Ownership Sequence Example.

The process steps are described below:

```
1) V1.create(M1)
2) V1.send(M1, 100)
3) Software purchase
4) M1.send(U1, 1)
5) S1.validate(U1)
```

where:

1. `V1` creates the `M1` "MasterAddress1" on the blockchain, representing a particular Software application.

2. `V1` adds 100 bitcoins to `M1`, loading it with some bitcoins that when transferred will represent entitlement to the Software application.

3. The end user purchases the Software application through a non-cryptocurrency transaction.

4. `V1` transfers a Master bitcoin from the `M1` address to the `U1` address. The transaction itself confers the ownership of the bitcoin, and the end user now has

the bitcoin from `M1`, the Master bitcoin, in the user's associated wallet. Hence, the user's ownership of a bitcoin from `M1` confers entitlement to the Software application, and is a transaction publicly verifiable on the blockchain.

5. The Software application then validates that `U1` has received a transaction from `M1`, and is the last transaction in the chain of transactions.

Ownership of the Master bitcoin can be transferred, as shown in Figure 4.3 on the preceding page, so that the software vendor can be guaranteed that only a single user is using the software through checking the blockchain "chain of title" for the Master bitcoin originating address. Hence, `U1` can now transfer ownership to a new party, `U2`. The transaction confers ownership, and any entity can trace the chain of transactions from `U2` to `U1` and back to `M1` to confirm that current ownership is held by `U2`.

```
M1.send(U1, 1)
U1.send(U2, 1)
```

Typically, bitcoins do not have serial numbers and, once a non-Master bitcoin is combined with a Master bitcoin, the originator of each specific bitcoin cannot be identified because the bitcoins have individual digital signatures that have been combined to create a new digital signature. However, in the MBM, the value of the Master bitcoin is not important: only that a transaction history from the originating Master bitcoin exists. This presents an additional challenge in a cryptocurrency ecosystem, where the Master bitcoin itself may be combined with non-Master bitcoins, which can then be used to create multiple subsequent transactions that originate from the same Master bitcoin address, thereby entitling all recipients in the transaction with a chain-of-title to the Master bitcoin address. As such, Fortin (2011) proposes that, if a Master bitcoin is combined with a non-Master bitcoin, then the recipient with the largest numeric value of the transaction in a "transfer of ownership" transaction is the owner. This characteristic is defined as the "non-divisible ownership" property of the MBM.

```
M1.send(U1, 1.0)
U1.send(U2, .4)
U1.send(U3, .6)
```

→ U3 has ownership.

Where a Master bitcoin is divided exactly in half, the ownership is decided by the first recipient of the transaction (Lebo, 2014).

```
M1.send(U1, 1.0)
U1.send(U2, .5)
U1.send(U3, .5)
```

→ U2 has ownership.

Although the MBM presents an opportunity for a chain-of-title based software licensing system, there are disadvantages that will deter use in the software industry. The MBM is essentially a binary system; the end user either has entitlement to run the software, or they don't. Although it is possible to create multiple Master bitcoins to represent individual software features for activation, this creates an overhead for each software feature, and there is no capability to include additional functions that are commonly used by software vendors, such as multi-user licensing for organisations, and management of corporate and private licenses. Furthermore, it is possible to quickly move Master bitcoins between end users, defeating the intent of the authorisation method that the MBM intended.

In summary, the entitlement of a user to claim a license to use a software application is conferred using a unique blockchain address to represent a particular software application. The MBM is used to provide immutable proof of ownership of a bitcoin

that originated from a specific address. However, the software application will need to be built with the capability to read the blockchain. Therefore we contend that the MBM is too simplistic in that: (i) it fails to take account of the complex nature of software licensing and distribution; (ii) that it introduces additional complexities into software development; and (iii) that it provides a means by which an attacker may intentionally gain access to the application executable and modify it.

4.3.2 ReSOLV Model

In this section, the ReSOLV Model is introduced. It builds on the MBM with the objective of addressing the principal weaknesses of the MBM.

The ReSOLV Model is notable for a tokenised approach to validation whilst still referring to a Master bitcoin. This approach offers improvements over the centralised software validation model, permitting the model to be applied to overcome the limitations outlined in the previous section. The Token in the ReSOLV Model is a digital signature that represents entitlement to a specific software application. A user address that holds a particular Token, from a unique vendor address, is entitled to the software license and therefore is entitled to use the software. The vendor/token combination represents the entitlement for use of the software.

Blockchain specifications vary across cryptocurrencies and, as such, cryptocurrencies can be designed and built with unique characteristics to meet specific applications. The ReSOLV Model uses a custom blockchain transaction specification that includes additional fields tailored to the requirements of a flexible SLV schema. This provides the scope needed for the wide range of users and license models in the modern technology environment. It can also provide several useful mechanisms using the blockchain as the basis for license validation, license upgrade, transfer of ownership, and even software integrity checking. A customised blockchain specification (Figure 4.4 on the next page)

may include new blockchain fields, to improve SLV, and to prevent software piracy, through software integrity checks and protecting the software from reverse engineering and executable code modification.

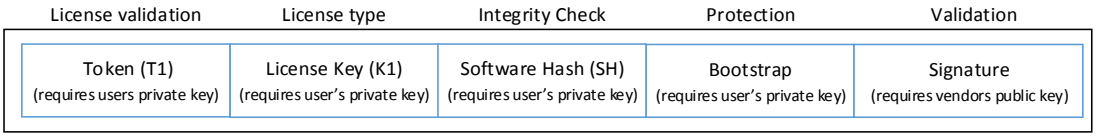


Figure 4.4: Customised Blockchain Specification for License Validation.

The fields in the customised blockchain (Figure 4.4) are stored as data and encrypted using cryptocurrency public/private key mechanisms, which are also built into the blockchain. The software vendor uses the user's public key to encrypt the data being placed into the fields, but the user's private key is subsequently required to decrypt the fields. The user can confirm the transaction integrity signature with the vendor's public key.

The custom fields outlined in the proposed specification are described as follows:

- *Token:* The Token is used for standard license validation mechanisms, where the ownership of the Token demonstrates entitlement. The Token can be used for SLV operations such as for software upgrades, or to provide a unique attribute to the transaction, such as "first 100 purchasers" that may have collectible value in the future.
- *License Key:* The License Key provides advantages over the MBM, because many software applications have specific features within the application that are activated on a per feature basis. Having the License Key securely held on the blockchain means software vendors can easily enable "feature activation", and have flexibility with software application licensing models, where users could rent software for short periods, rather than purchasing or renting use on a long term or monthly basis.

- *Software Hash:* Similarly, the vendor can place a software hash of the application on the blockchain. A bootstrap loader, or the software itself, can read the hash and check the software version. This hash can be updated with every new patch, plus minor or major releases of the software. This is intended to protect software from malware infection and some forms of reverse engineering.
- *Bootstrap Loader:* Additional protection may be provided through a bootstrap loader, a portion of executable code used to pre-execute the software application or as an integral part of application execution. The purpose of the bootstrap loader is to further prevent reverse engineering of the software application. When the unencrypted bootstrap code is executed and stored in memory, it may be susceptible to interception. Therefore the bootstrap code can change with every patch and minor and major release, making reverse engineering of code a constant effort.
- *Signature Field:* The signature field is a possible additional field that can be used by the vendor to sign the entire transaction content using the private/public key pair of the software MasterAddress.

Existing software validation uses digital signatures to verify downloadable software, and digital certificates to prevent Man-in-the-Middle attacks during the download process. However, the proposed custom specification provides validation of installed software on the user's device on an ongoing basis, providing risk mitigation against malware code injection attacks.

Compared with the MBM, the ReSOLV Model presents significantly more opportunities to use the Token for SLV. In addition to validating that the user owns the Token and verifying rights to upgrade software versions or transfer of ownership, the Token may also be used to validate rights while the software is running or at critical moments. For example, it could be used for reading the Token at a regular time interval, at user

login/logoff, or at software start-up or shutdown. These examples read an existing transaction on the blockchain but don't create a transaction.

For updating information on the blockchain (see below), a transaction that includes the user and software addresses that represents the software application is created. Each transaction creates a new user address, with its own public/private key pair and with data encrypted by the user's new public key. Note that all addresses are unique, with their own public/private key pairs.

```
1) S1.read(UserAddress1.transaction)
2) S1.decrypt(UserAddress1.Token)
3) S1.validate(Token)
4) S1.execute
```

where:

1. `S1` reads the blockchain transaction for `U1`.
2. `S1` decrypts the Token from blockchain data for `U1`.
3. `S1` checks the Token originates from `M1`.
4. `S1` continues to execute on the end user's device.

To upgrade software application versions, for example by applying a service patch update (see below), the software application periodically requests an update from the vendor. The new `U2` address for the upgraded software application provides a record of entitlement to earlier software versions is maintained through `U1`. A practical benefit of maintaining a previous version allows the potential to roll back an upgrade. All `Ux` addresses are stored in the user's digital wallet and, as such, all entitlements are associated with the user. To reduce the risk of license key duplication, the vendor may release new license keys with minor releases such as patch updates.

```
1) S1.send(MasterAddress1, UserAddress2, Token)
2) M1.validate(UserAddress1, Token)
3) M1.send(UserAddress2, Token)
4) S1.read(UserAddress2, Token, License, Hash)
5) S1.upgrade
```

where:

1. `S1` sends a request to `M1` with new `U2` address.
2. `M1` checks the Token came from `U1` and is valid.
3. `M1` creates a new transaction with update data.
4. `S1` reads data to check if it needs an upgrade.
5. `S1` auto-upgrades.

Although similar to existing online software version checking mechanisms such as Microsoft Update, the process shown below provides the opportunity to "re-cut" a license key or hash code for the software upgrade and have the software automatically validated. The process below reframes the transaction process shown in Figure 4.5 on page 129 to include the additional fields. Both the previous software version and the upgraded software version are subsequently available for use.

```
1)  S1.send(MasterAddress1, UserAddress2, Token)
2)  M1.validate(UserAddress1, Token)
3)  M1.createaddress(M2)
4)  M2.License(License.new)
5)  M2.hash(S1.new)
6)  M2.bootstrap(Bootstrap.new)
7)  M2.encrypt(M2.License)
8)  M2.encrypt(M2.hash)
```

```
9)  M2.encrypt(Bootstrap.new)
10) M2.sign(Transaction.new)
11) M2.send(UserAddress2, Token)
12) S1.read(UserAddress2, Token, License, Hash)
13) S1.upgrade
14) S1.execute
```

where:

1. `S1` sends an update request to `M1` with new `U2` address.
2. `M1` checks Token came from `U1` and is valid.
3. "MasterAddress2", `M2`, created for the new transaction.
4. `M2` cuts new License Key for new software version.
5. `M2` creates Hash for new software version.
6. `M2` creates new Bootstrap for new software version.
7. `M2` encrypts the new License Key, Hash and Bootstrap using `PublicKey(U2)`.
8. `M2` signs the transaction with `PrivateKey(M2)`.
9. A new process is started.
10. `M2` creates new transaction with the new data.
11. `M2` sends `U2` Token with update data.
12. `S1` reads new transaction data for upgrade.
13. `S1` downloads software and auto-upgrades.
14. `S1` runs itself.

In summary, the ReSOLV model demonstrates that license validation can be achieved with the application of the blockchain and, through the same mechanism, integrity and security protections can be added. Furthermore, blockchain smart contracts allow for intelligent programming of actions within a transaction. This provides a

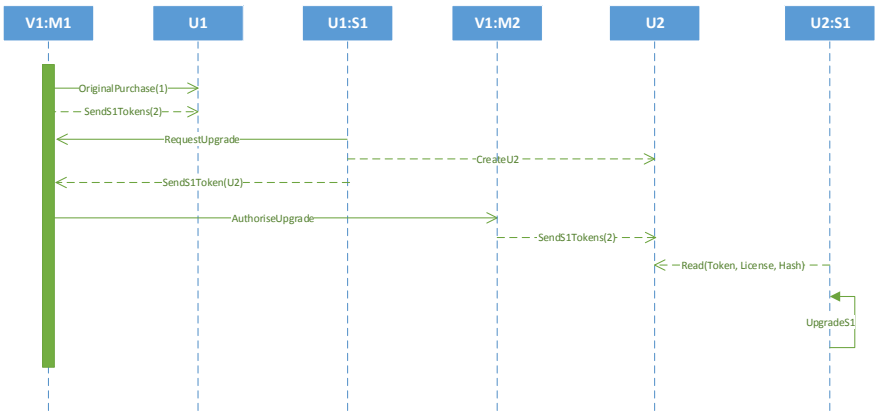


Figure 4.5: ReSOLV Blockchain Software Upgrade Sequence Example.

new level of dynamism as a transaction may take different actions based on the input, output, field content, and originating and destination address. Blockchain protocols, such as Ethereum, that provide Turing completeness capability allow the developer to write smart contracts and decentralised applications, with arbitrary rules for ownership, transaction format and state transition functions (Wood, 2016).

4.3.3 Issues Overcome

While the ReSOLV Model overcomes problems noted earlier, and significantly improves on the MBM, it does require a separate cryptocurrency ecosystem to be developed and maintained. The MBM can utilise, and run in, an existing cryptocurrency ecosystem and meets the requirements outlined for a successful SLV method. The MBM has been demonstrated in a proof-of-concept, but the model also presented limitations that may detract from its usefulness. The ReSOLV Model overcomes these limitations:

1. Each software license is hard to copy because the license is represented by a transaction between vendor and user, is cryptographically verifiable, and is stored in the user's blockchain wallet. An adversary would require the password to the user's wallet in order to access the user's private key. Already, multi-factor authentication mechanisms are available to further enhance user wallet security.

In addition, the license key does not have to be disclosed to the user, so the key cannot be copied. Every transaction is cryptographically secure and cannot be modified.

2. Software licenses are easily validated, through the blockchain "chain of title" and the data being held within the blockchain itself. Furthermore, having on-blockchain license keys allows the vendor to distribute keys for specific feature activations, and allows keys to be "re-cut" quickly and efficiently without the need for an intermediary.
3. Software licenses cannot be regenerated because the software application takes the license key directly from the blockchain, requiring the user's private key. Even if the key generator at the vendor is compromised, there is no way to get the license key onto the blockchain without the vendor's private key to sign the transaction.
4. No Man-in-the-Middle attack is possible, using the blockchain. An adversary cannot make use of any intercepted data in the blockchain, without the user's private key. Also, an adversary cannot redirect DNS or IP traffic to an adversary's custom server to provide software validation, without the vendor's private key.

Additionally, the peer-to-peer blockchain architecture means that no single central point of failure for SLV exists. Licensing validators can be run anywhere and, for example, could be run either commercially or on a not-for-profit basis. In particular, a vendor may run vendor-specific software to manage a license creation process and interaction with the blockchain, but does not need to maintain a dedicated license validation infrastructure with its associated overheads.

Therefore, the ReSOLV Model provides an opportunity for small to large software vendors to preserve software copyright and prevent software piracy, whilst having a flexible mechanism for software licensing.

4.3.4 Summary

In this section, the requirements for the blockchain-based SLV have been discussed, building on the MBM. The ReSOLV model was discussed, with transaction models demonstrating how a successful token transfer would occur in blockchain-based SLV. The ReSOLV Model provides several improvements over the MBM, but also requires further consideration of potential issues that are likely to occur, due to the complex actor interaction and scalability of the blockchain ecosystem.

The following section discusses the requirements specification, introducing the ReSOLV HLA, an overview of an expanded blockchain-based SLV ecosystem, and the ReSOLV RA, which forms the core functions of ReSOLV. Functional and Non-functional requirements are also established and discussed.

4.4 Requirements Specification

4.4.1 Introduction

This section discusses the requirements specification for the ReSOLV SLV ecosystem. The ReSOLV HLA and ReSOLV RA for functional requirements are introduced. Non-functional requirements, and public key cryptography, which lies at the heart of the core functions of ReSOLV, are also discussed for completeness. Producing a functional prototype artefact is beyond the scope of this research. The requirements specification provides an architecture and design perspective to describe the methods for SLV. The requirements specification commences with a mission statement, the highest level of a Goal Breakdown Structure, which is a hierarchical structure linking high-level objectives or goals to more detailed goals.

Laplante (2014) states that the first requirement for development of a new system, is to develop a concise description of what it is supposed to do. The product mission

statement acts as a focal point for all involved in the system, and it allows the stakeholders to prioritise features by asking the question, "How does that functionality serve its intent?". The Mission Statement for RE is described as follows:

Mission Statement

"The purpose of ReSOLV is to provide software piracy prevention and provenance (SPPP), using blockchain technology to validate user license entitlements, and to validate software integrity."

4.4.2 ReSOLV High Level Architecture

In this section, the ReSOLV HLA is introduced, expanding the ReSOLV Model into an enlarged blockchain ecosystem of actors, functions, and services. The HLA builds on the principles described in the ReSOLV model, proposing core actors that form the basis of the ReSOLV RA. Figure 4.6 on the following page, provides an overview of the architecture and vision for the ReSOLV blockchain ecosystem, encompassing SLV for single users, enterprise environments, SaaS providers, and supporting Internet services.

Three Primary Actors (PA) of the ReSOLV HLA are defined, namely the Vendor (PA1), User (PA2), and ReSOLV Corp (PA3). The high level functions of these PA's are described below.

PA1: Vendor

There are three main functional requirements that are needed for the Vendor in the ReSOLV ecosystem: a provenance agent/service, a software encoding service, and the vendor wallet.

Provenance Agent/service: provides the front-end interactions with the blockchain Users and ReSOLV Corp. These include: (i) receiving notification of a software purchase from an external mechanism; (ii) receiving a license token from the

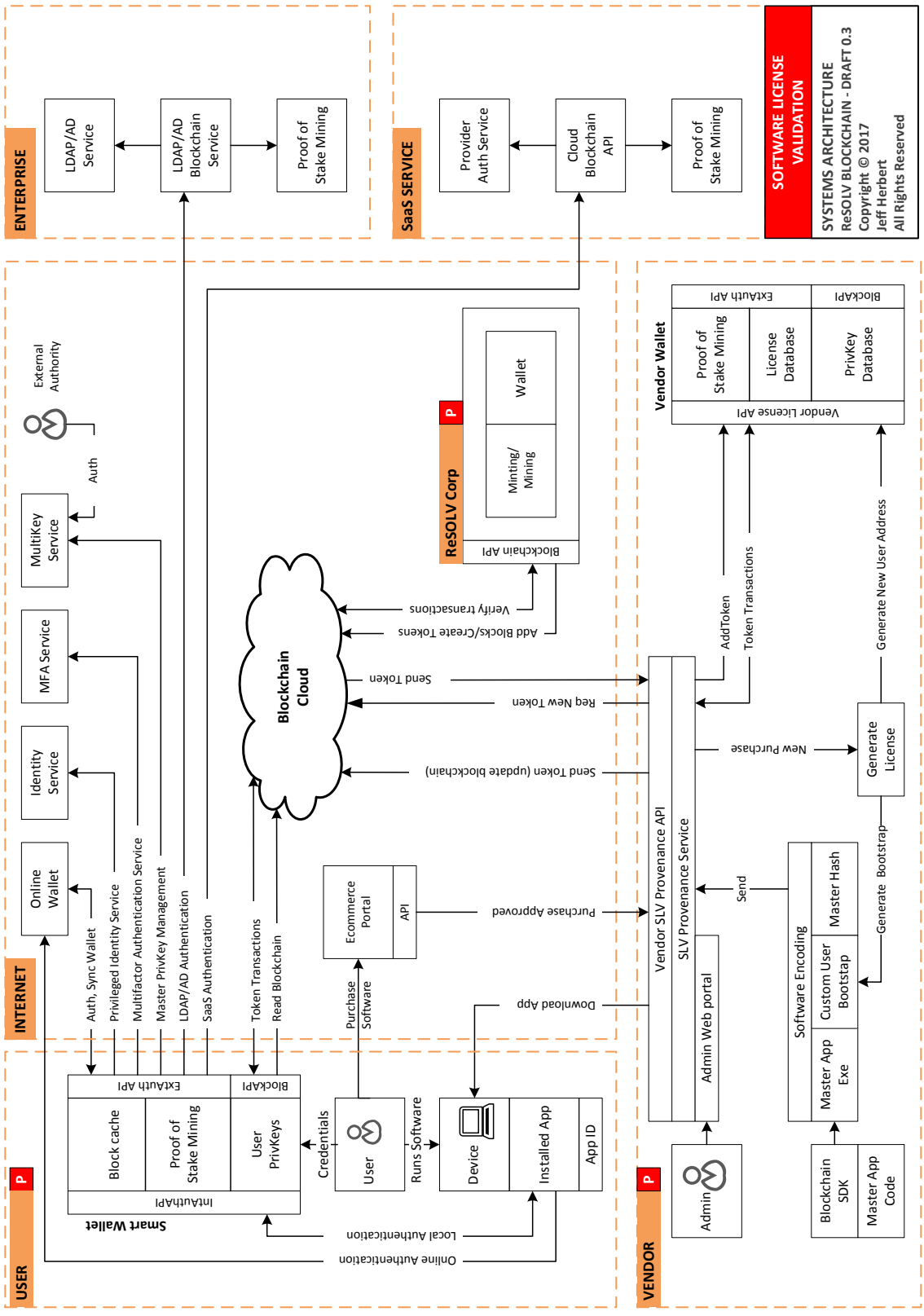


Figure 4.6: Proposed ReSOLV HLA

Software Encoding Service; (iii) sending a license token to the purchaser (the user); (iv) receiving tokens from the ReSOLV Corp mining/minting function (purchasing tokens); and (iv) receiving software encoded with a wrapper that provides the user authentication method. This software can be published through normal mechanisms or provided to the user as a download.

Software Encoding Service: provides the back-end software encoding and licensing services. These include: (i) receiving a license key from the Vendor Wallet, which defines the user entitlements for the purchased software; (ii) digitally encrypting the license with the users public key, which is subsequently published to the blockchain, thereby making the key publicly available; (iii) constructing metadata, such as a software bootstrap and master hash, which are also published onto the blockchain with the license key and are used by the SmartWallet to validate the integrity of the software installed on the user's computer; (iv) encoding software with a wrapper to provide user authentication prior to executing the software, which is used by the SmartWallet and App to validate the user.

Vendor Wallet: provides two functions: a secure database function, and a mining function. The secure database contains a list of keys for license entitlements, user public keys that each license has been allocated for, and a unique private key for each license, which is used throughout the lifecycle of the license. The vendor wallet participates in the blockchain ecosystem by providing mining functions to validate transactions, and also includes the possibility of minting new coins, (creating free tokens, thereby incentivising the vendor).

PA2: User

The User actor requires two main functions in the ReSOLV ecosystem: the SmartWallet, and the encoded software.

SmartWallet: is an application that is a service that is part secure database, part

cache, and part miner, and provides the front-end interactions with the blockchain externally, as well as internal functions, servicing requests from software for license validation. The SmartWallet functions include: (i) securely storing all the private keys of the user; (ii) caching specified blockchain blocks that contain the user's license validation metadata, so that licenses can be validated offline; and (iii) participating in the blockchain ecosystem, providing Proof-of-Stake mining/minting and transaction validation, and similarly to the vendor, includes the possibility of minting new coins (creating free tokens which may be used to purchase software, thereby incentivising the User).

Encoded Software: is downloaded from the Vendor, requires an authentication mechanism to the SmartWallet in order to validate user entitlements to run the software.

PA3: ReSOLV Corp

ReSOLV Corp is the actor that has the primary off-blockchain role of supporting the on-going development of the ReSOLV ecosystem, and providing a governance framework required to mitigate risks associated with the blockchain and increase value to the ecosystem actors. The on-blockchain functions are limited to:

Minting: ReSOLV Corp provides the initial blockchain genesis block, minting of the initial tokens, and managing the token supply to all the Vendors, using its own wallet to store the tokens.

Mining: ReSOLV Corp also provides blockchain ecosystem mining functions on an ongoing basis, as the most trusted actor on the blockchain.

The HLA also extends the architecture to include: (i) Enterprises, where integration into Lightweight Directory Access Protocol (LDAP) or Active Directory (AD) services is required for corporate user identity validation; (ii) SaaS services, which may potentially prefer a blockchain authentication method over a centralised OAuth[3] type

[3] https://oauth.net/

service; and (iii) Supporting Internet actors that may include proof of identity, multiple signature support, and online wallets secured by hardware wallets such as the Trezor[4] and LedgerWallet[5].

4.4.3 ReSOLV Reference Architecture

This section describes the ReSOLV RA as shown in Figure 4.7 on the next page. The RA shows the human actors as the starting point to describe the primary processes and methods of the ReSOLV blockchain ecosystem. It also shows the related technology actors such as mining agents, wallet agents, encoding service, and provenance agents. Primary processes, which are related to the human actors, group the ReSOLV Methods and are shown using colour coding of the Methods. The ReSOLV Methods themselves demonstrate the flows of data in the ReSOLV blockchain ecosystem (described below), and are labelled according the human actor (U=User, V=Vendor, C=Corp).

Similar to the lack of research methodologies for new cryptocurrencies, RAs for cryptocurrencies tend to be limited to white papers. However, there are examples of RAs available. Wood (2016) published a RA for Ethereum proof-of-work method, whilst Alqassem and Svetinovic (2015) proposed a Bitcoin RA focussing on the fundamental methods for the Bitcoin protocol. For the purpose of this research, the ReSOLV RA provides an overview of the fundamental actors and methods for SLV, but does not provide the details for the entire ecosystem. This allows the focus to be on the SLV process itself, rather than the supporting blockchain ecosystem methods (which are vital in the distributed system, but not the core aspect of this research).

Human Actors

There are three human actors described in the ReSOLV blockchain ecosystem: “Corp”, “User”, and “Vendor”. Corp represents the ReSOLV corporation physical entity that

[4] https://trezor.io/
[5] https://www.ledgerwallet.com/products

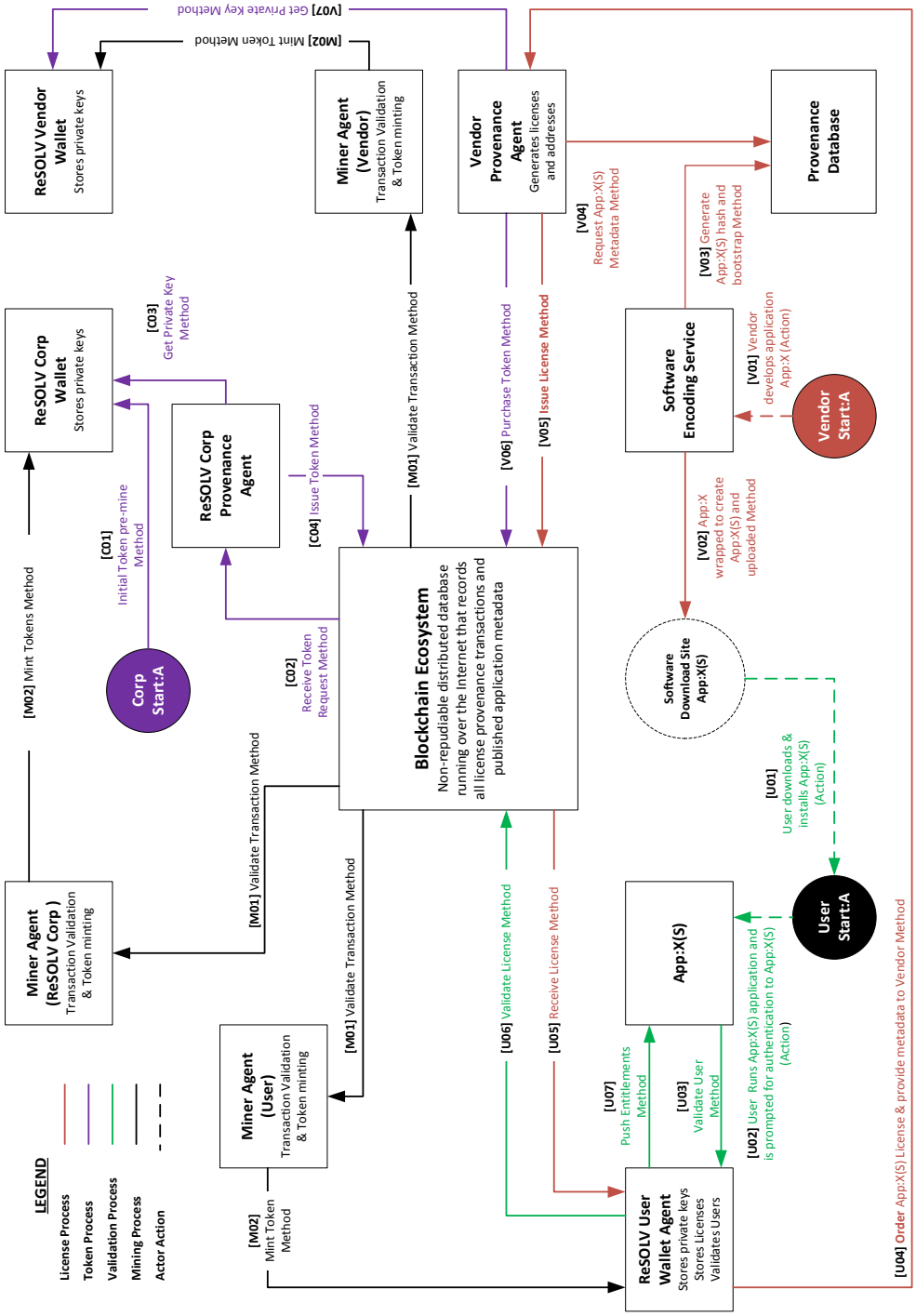


Figure 4.7: The ReSOLV RA

is responsible for the creation, development, governance, and ongoing operation of the ReSOLV blockchain ecosystem. The Vendor is the software company that has developed an application that will be protected using ReSOLV, whilst the User is the consumer that downloads, installs and runs the application from the Vendor. Note that each of the human actors has their own Wallet Agent to store tokens and private keys.

Primary Processes

The primary processes provide a visualisation of how the methods are related in the ecosystem. The Token processes (shown in purple) are related to the creation and transfer of unserialised tokens: that is, tokens that have not been distributed to a User with a license key. License processes (shown in brown) are associated with the encoding of the Vendor application and license generation and issuance, whilst Validation processes (shown in green) are the processes surrounding user execution, authentication, and license validation. Actor actions are events initiated by the actor and are shown as dashed lines.

Technology Actors

Technology actors include the provenance agents, mining agents, user wallet agent, and the software encoding service. The ReSOLV Corp provenance agent is responsible for token issuance to vendors, whilst the Vendor provenance agent is responsible for license issuance to the User. The Vendor also has a Software Encoding Service, which is responsible for wrapping the application with an executable wrapper that enforces user authentication, when executed, to determine license entitlements. For the purpose of the research objectives, there is no actual encryption of the application being undertaken within the wrapper. This extension of protecting the application could be implemented as a future requirement. The Mining agents participate in the transaction validation and token minting processes, with newly minted tokens being stored in each actor's wallet. Implementing this concept provides an economic incentive for Vendor and User actors to create tokens for use within the ecosystem. Although these methods are essential for

the operational blockchain ecosystem, they are not fundamental to the ReSOLV process, and are not defined in the requirements specification.

4.4.4 Methods

Following on from the previous section, the methods for the ReSOLV RA are explained briefly below, illustrating the activities between the Agent actors.

 # CORP

C01: Initial Token Pre-mine Method: the blockchain ecosystem will require an initial pre-mine of tokens to get the system running and commence the minting of new blocks. Pre-mined tokens are stored in the ReSOLV Corp wallet.

C02: Receive Token Request Method: vendors must request (purchase) tokens from the ReSOLV Corp Provenance Agent for use in the license issuance process.

C03: Get Private Key Method: the ReSOLV Corp Provenance Agent requests the required private key from the Corp Wallet.

C04: Issue Token Method: the ReSOLV Corp Provenance Agent issues the required Tokens to the Vendor Wallet Agent.

 # VENDOR

V01: Vendor develops application App:X (Action): this is the initiating trigger for the vendor, who develops application App:X and submits it to the Software Encoding Service.

V02: App:X wrapped to Create App:X(S) Method: App:X is "wrapped" with an executable that provides the front-end authentication to establish entitlements and execute the application. For the purposes of testing the ReSOLV method, this can be a stand-alone application that calls the executable software.

V03: Generate App:X(S) and bootstrap Method: the hash of App:X(S), and the bootstrap, are stored in the Provenance database. These are used by the ReSOLV

user Wallet Agent to validate that App:X(S) has not been tampered with, and to provide the code segment that will allow the App to boot.

V04: Request App:X metadata Method: when the User orders a license for App:X(S), the Vendor provenance agent triggers a request for App:X(S) hash and bootstrap.

V05: Issue license Method: the Vendor provenance agent generates the license requested by the User, and a unique blockchain address for the license. It then creates a Sidebar, which is defined as "a set of data fields containing the license key, hash and bootstrap, all encrypted with the User's Public key, and signed with the Vendor private key." It then places the Sidebar on the blockchain from the unique address.

V06: Purchase Token Method: the Vendor provenance agent requires tokens to which the Sidebar is attached. The Vendor must acquire the tokens from the ReSOLV Corp (or have minted tokens through mining). This method will trigger when the number of tokens in the Vendor Wallet is low or nil.

V07: Get Private Key Method: when the Vendor provenance agent requires a private key for a ReSOLV address from the Vendor Wallet for a software license, it receives the next available private key.

 # USER

U01: User downloads and installs App:X(S) (Action): the User downloads and installs App:X(S). This is the initiating trigger action by the User.

U02: User runs App:X(S): the User executes the App:X(S) and is prompted for authentication by the App:x(S) wrapper.

U03: Validate User Method: App:X(S) sends the username and password hash to the User Wallet Agent via an ephemeral port or through a secure Application Programming Interface (API).

U04: Order App:X(S) license Method: the User Wallet Agent authenticates the credentials

against those entered into its database. If there is no entry in the database, the User Wallet Agent sends an order for an App:X(S) license to the Vendor Provenance Agent.

U05: Receive License Method: the User Wallet Agent receives a token representing the purchase of App:X(S) from the Vendor.

U06: Validate License Method: the User Wallet Agent validates the license by reading Sidebar from the blockchain, using the Vendor address and the User's private key.

The following sections discuss the non-functional requirements for the ReSOLV blockchain ecosystem, and introduce public key cryptography, which is at the heart of blockchains, cryptocurrencies, and ReSOLV itself.

4.4.5 Non-functional Requirements

The non-functional requirements defined in this section are focussed on blockchain requirements rather than SLV requirements, as the blockchain is the underlying platform that ReSOLV will run on. These requirements are established for completeness of specifying requirements of the ReSOLV blockchain ecosystem. Each requirement in itself requires significant investigation and, as such, these non-functional requirements are out of the scope of this research.

The requirements are established through the following Related Works: (i) SLV in Section 2.7 on page 85; and (ii) alternative blockchain applications in Section 2.6 on page 77. Building on these requirements, the non-functional requirements in the ReSOLV blockchain ecosystem are considered from the following common perspectives: (i) Design/architecture; (ii) Usability; and (iii) Governance Considerations. These requirements are explained below, with each requirement being described succinctly to demonstrate the purpose of each characteristic for the ReSOLV blockchain ecosystem.

There has been diverse industry discussion on these requirements through various discussion forums, and the consensus is that these types of non-functional requirements need to be addressed during the design phase of the cryptocurrency or blockchain. Such problems are currently (as at the date of submission) being seen in the Bitcoin design stand-off between miners and developers, arguing over how to improve the Bitcoin protocol scalability, which was not factored into the original Bitcoin design. These non-functional requirements also provide the impetus for the creation of new cryptocurrencies, collectively known as distributed ledger systems, such as Dash[6], and blockchain alternatives such as hashgraph (Baird, 2016).

- **Design Requirements**

  Scalability: where blockchain design needs to consider many parameters surrounding long-term scalability, including transactions per second, block size, blockchain bloat, validation period, capability to handle micro-transactions, mining requirements (Wood, Zamfir & Coleman, 2015; Danezis & Meiklejohn, 2015; Oberhauser, 2015).

  Robustness: reflecting the need for the blockchain ecosystem to be hard to disrupt through bugs, growth, delays in transaction validation, changing environment, or malicious attacks on the blockchain miners and protocols (Mazières, 2015; Kondor, Pósfai, Csabai & Vattay, 2014; Glaser & Bezzenberger, 2015).

  Resiliency: the ability to be able to provide continuity of operation if an unexpected event occurs, or to recover to full operation without manual intervention or corrective actions required (Bott & Milkau, 2016; Glaser & Bezzenberger, 2015).

  Data Integrity: to ensure full data integrity of the blockchain through proven

[6] https://www.dash.org

and trusted cryptographic methods (Alqassem & Svetinovic, 2015; European Central Bank, 2015).

Security: the blockchain ecosystem can demonstrate that the bitcoin transaction are secure throughout the transaction and consensus process, ensuring double-spending does not occur. This includes ensuring the blockchain proofs are valid. (Miers, Garman, Green & Rubin, 2013; Meiklejohn et al., 2013; Xiaochao, 2014).

- **Usability Requirements**

  From a ReSOLV user perspective, several non-functional requirements must be considered to ensure the ReSOLV method will continue to function, given the varying environments in which a user may be consuming the software. Typical usability requirements are defined as follows:

  Platform Independence: where the User may be using the software on different devices (e.g., a Windows PC at home and a Windows PC at a Net cafe).

  Mobility: where the User may be using the software across different geographic locations (e.g., using the software on a device whilst in an office or using it on the same device whilst on a ship at sea).

  Portability: where the User may be consuming the software on different platforms (e.g. the user wants to play a game on a PC or a PlayStation 4).

  Flexibility: where different license types can be assigned to the user depending on their need (e.g. where the vendor licenses specific feature sets in software, which the User can select, or licenses can be revoked or expire).

  Standalone: where the user can use the software without being connected to the Internet (e.g. where Internet is not available, such as, desert region).

- **Governance Considerations**

Cryptocurrency and blockchain technologies introduce a new dynamic of governance over the ecosystem, which should be established during the development process phase and to provide on-going direction (Peck, 2017; Millar, 2017).

Hostile Takeover: controls are required to prevent hostile takeover of the cryptocurrency or blockchain by the Miners, or pools of Miners, forming 51% collective mining power (Buterin, 2014; Ali et al., 2016).

Ownership: controls are required to ensure the funding entity retains ownership of the cryptocurrency or blockchain development once it has reached go-live (NXT Community, 2014; Millar, 2017).

Stability: methods are required to ensure Miners are mining the blockchain using the latest stable release of mining software (Millar, 2017).

Economic Design: decisions need to be made as to what economic parameters will be incorporated into the cryptocurrency or blockchain design to ensure sustainable functionality (Millar, 2017; Fry & Cheah, 2016).

Code Source: policies need to be established to determine whether existing source code is used, private source code is released, or if the development is closed-sourced (Martin, 2014; Millar, 2017).

4.4.6 Public Key Cryptography and Digital Signatures

Cryptocurrencies and blockchain technology use a range of cryptographic primitives to enable the decentralised trust and the immutability of the blockchain ledger. There are two primary cryptographic functions used by cryptocurrencies: (a) public key cryptography to store and spend money and; (b) cryptographic validation of transactions using digital signatures (Böhme, Christin, Edelman & Moore, 2015).

Public Key Cryptography

Public key cryptography is a form of asymmetric cryptography, which uses key pairs (a public key and private key) to provide authentication and encryption of data (messages) that are exchanged between two parties over an insecure medium (such as the Internet). The public key is a public identifier that can be freely shared with others, which is used by others to encrypt data being sent to the owner of the public key. The private key is a secret or password that must never be shared with anyone, and is used by the owner to decrypt received messages that have been encrypted with the public key (Rivest, Shamir & Adleman, 1978; Boneh, Sahai & Waters, 2012). For example, when Alice wants to send a message securely to Bob over the Internet, she uses Bob's public key to encrypt the message and then sends it to Bob. When Bob receives the encrypted message from Alice, he uses his private key to decrypt the message. If Bob wants to reply to Alice, he uses Alice's public key to encrypt the message and then sends it to Alice, who in turn, uses her private key to decrypt the message.

Unlike symmetric key cryptography, public key cryptography does not require a secure exchange of secret keys and functions securely based on full disclosure of the public key. Public key cryptography relies on cryptographic algorithms based on mathematical problems. Cryptocurrencies primarily utilise elliptic curve cryptography (ECC), which provides for fast, efficient computations with low key size and high security (Singh et al., 2016; Wood, 2016).

Digital Signatures

The second cryptographic function used by cryptocurrencies, digital signatures, also uses public key cryptography. Digital signatures mathematically prove authenticity that a message received is really from the person expected, and that the message contents have not been modified during transmission over the insecure medium (Rivest et al., 1978). The sender "signs" the message, and the output value of the "sign" function is called the digital signature. Even a single character difference in a message will result

in an entirely different digital signature.

As shown below, Alice may input a `Hello World` message, along with her private key (priv_key) through a sign function, to create a unique digital signature. If a single character such as an "!" is added, the digital signature is completely different.

Original message:

```
sign("Hello World", priv_key) = n67n54n6l10xf15
```

Modified message:

```
sign("Hello World!", priv_key) = vk34jxl140501025
```

Bob, the recipient of the message from Alice, can use Alice's public key to verify the message digital signature to guarantee both the integrity of the message and that the message was sent by Alice. If a single character has been changed and a man-in-the-middle or forgery attack was attempted by a threat actor, the digital signature, validated using Alice's public key, would not match, and Bob would know the message integrity was compromised.

Public key cryptography and digital signatures are essential for data integrity and authenticity (non-repudiation in a legal sense) in the transmission of secure messages over insecure, and indeed hostile, mediums, such as the Internet. Public usage of both cryptographic functions has increased, as innovation over the Internet drives new use-cases, such as cryptocurrency and blockchain applications.

As Applied to Cryptocurrencies

Cryptocurrencies use public-key cryptography to validate transactions between all participants, and digital signatures to ensure transactional integrity and non-repudiation (Peteanu, 2014). The cryptographic mechanisms used by cryptocurrencies provide for strong confidentiality, data integrity, and non-repudiation services that are in use by business, government, and military organisations, globally. In a cryptocurrency ecosystem, the public key can be considered as the participant's account number, whilst

the private key represents the participant's ownership credentials. All participants have digital wallets that are used to store private keys, as well as digital signatures that represent cryptocurrency entitlements (bitcoins) that the participants own.

As Applied to ReSOLV

ReSOLV utilises public key cryptography and digital signatures, as applied to cryptocurrencies, for the transfer of the token (the digital signature) between accounts (user wallets). It also utilises public key cryptography for encryption of the Sidebar data, namely the software license, the hash of the software, and the software bootstrap. This process maintains confidentiality and integrity of the data that resides on the blockchain. ReSOLV also utilises digital signatures to provide authenticity of the Sidebar itself, and places the Sidebar digital signature on the blockchain, so that any party can validate that the Sidebar is from the expected software vendor.

Each software item, shown as App:X in the RA in Section 4.7 on page 137, requires a new public-private key pair that will be used to store the digital signature for the license in the User Wallet Agent. The User Wallet Agent, as the master repository of all the private keys, will also need to be strongly protected itself, as a compromise of the wallet will expose all the private keys, to decrypt the software Sidebar and access software license information.

4.5 ReSOLV User Stories

The following User Stories reflect the functional requirements for the human and technology actors identified in the ReSOLV RA. The User Stories have been structured as described below.

- ReSOLV Corp user stories are established in Table 4.1 on page 149.
- ReSOLV User user stories are established in Table 4.2 on page 150.
- ReSOLV Vendor user stories are established in Table 4.3 on page 151.

Notes:

- [U03] Validate User Method user stories have been deconstructed to reflect the different resulting states that may result from the [U02] user authentication, such as a failed authentication or not having a license for the application, App:X(S).
- [U06] Validate License Method reflects user stories for both successful and failed validation of App:X(S).
- App:X(S) is used to denote that Application X has been wrapped with the SLV wrapper that enables authentication by the user.

4.6 Summary

In this chapter the RE discipline and process was discussed and applied to the ReSOLV SLV method. The requirements elicitation process built on the MBM, identified in the literature, to establish the requirements for the ReSOLV model blockchain-based SLV method. To further define the ReSOLV model requirements, a requirements specification is established. The requirements specification introduces a ReSOLV HLA that outlines the entire ReSOLV blockchain ecosystem. It also introduces the ReSOLV RA that describes the core methods for ReSOLV SLV, from which User Stories are established. Non-functional requirements are discussed for completeness, but are out of the scope of this research.

In the next chapter, a FD of key ReSOLV methods is presented, and DFDs are introduced with minispec examples in pseudocode, which describe processes in more detail. The DFDs provides the fundamental description of the core ReSOLV processes.

Table 4.1: ReSOLV Corp – User Stories

Ref	GIVEN	WHEN	THEN
#C01	that the ReSOLV genesis block has not been created	the Corp Miner is first executed	generate the genesis block with pre-mined tokens
#C02	that a token is requested by a Vendor Provenance Agent	at any time	send a token request to the Corp Wallet agent with the required number of Tokens
#C03	that the Corp Wallet agent receives a token request from the Provenance Agent	at any time	generate a new blockchain address and provide the required quantity of tokens to the ReSOLV Corp Provenance Agent using the new address
#C04	that the Corp Provenance agent has a new ReSOLV blockchain address generated	at any time	update the blockchain with the new address and tokens

Table 4.2: ReSOLV User – User Stories

Ref	GIVEN	WHEN	THEN
#U01	that the user downloads App:X(S)	the user installs App:X(S) on the user device	App:X(S) is ready to run
#U02	that the user runs App:X(S) and the App:X(S) wrapper presents an authentication window	the User enters their credentials	the App:X(S) wrapper will push the hashed User credentials and App:X(S) details to the User Wallet Agent to validate the User
#U03A	the hashed User credentials received by the User Wallet are successfully validated against the database	App:X(S) entitlements are successfully validated in the database	push App:X(S) entitlements to App:X(S)
#U03B	the hashed credentials received by the User Wallet are unsuccessfully validated against the database	at any time	push App:X(S) invalid credential message to App:X(S)
#U03C	the hashed credentials received by the User Wallet are successfully validated against the database	App:X(S) entitlements are found in the database but are invalid	push App:X(S) invalid entitlement message to App:X(S)
#U04	the hashed credentials received by the User Wallet are successfully validated against the database	App:X(S) entitlements are not found in the wallet database	order a license for App:X(S) from the Vendor Provenance Agent and provide the metadata including the App:X(S) identifier, User name, User Wallet address & the Wallet address public key
#U05	that an order exists for a license for App:X(S) from the Vendor Provenance Agent	each 15 seconds has elapsed	query the User Wallet address on blockchain for the new App:X(S) Sidebar
#U06A	that license App:X(S) exists and a minimum of 24 hours from the previous App:X(S) validation has elapsed	the agent connects to blockchain to read license Sidebar and successfully validates App:X(S)	update database validation log for a successful validation check
#U06B	that license App:X(S) exists and a minimum of 24 hours from the previous App:X(S) validation has elapsed	the agent connects to blockchain to read license Sidebar and unsuccessfully validates App:X(S)	update database validation log for an unsuccessful validation check and alert the User

Table 4.3: ReSOLV Vendor Provenance Agent User Stories

Ref	GIVEN	WHEN	THEN
#V01	that the vendor develops App:X	App:X is uploaded into the Software Encoding Service	App:X is ready for wrapping
#V02	App:X is ready for wrapping	App:X has the ReSOLV wrapper applied by the Software Encoding Service to become App:X(S)	then the Vendor can upload App:X(S) to an external software repository or portal for download by any public user
#V03	App:X has the ReSOLV wrapper applied to become App:X(S)	the App:X(S) hash and bootstrap has been generated by the Software Encoding Service	store the App:X hash and bootstrap in the Provenance Database
#V04	the User Wallet Agent has ordered a license for App:X(S)	when the App:X(S) hash and bootstrap have been read from the Provenance Database	generate a new license address, license key and create the App:X(S) Sidebar. The Sidebar consists of AppX:(S) license key, hash & bootstrap all encrypted with the User public key, plus the hash of the previous three artefacts digitally signed with the Vendor private key.
#V05	the App:X(S) Sidebar has been created	at any time	update the blockchain with the App:X(S) Sidebar
#V06	that the Vendor Wallet has less than ten Tokens	when a Token is requested from the Vendor Wallet	purchase ten Tokens from the Corp Provenance Agent through the ReSOLV blockchain ecosystem
#V07	that the Vendor Provenance Agent requires a new address for a software license	an order for a license for App:X(S) is approved	generate a new address from the next available private key in the Vendor Wallet

Chapter 5

Functional Decomposition

5.1 Introduction

In the previous chapter, the RE approach for establishing the ReSOLV method requirements was discussed. The ReSOLV blockchain-based SLV model was proposed, and a HLA and RA established and described. RE focusses on the functional requirements for ReSOLV, resulting in user stories for the RA core processes being created as the primary output from the RE research activities. Consideration was also given to the non-functional requirements for ReSOLV. Although these are out-of-scope of this research, they provide a useful overview which establishes the characteristics required to achieve the ReSOLV mission statement.

In this chapter, an FD of the ReSOLV RA is undertaken to identify what processes need to be built. DFDs are introduced for two of the main systems established in the ReSOLV RA (see Section 4.7 on page 137) as follows: (i) the Vendor Provenance System (VPS), relating to the Vendor Provenance Agent; and (ii) the Client User System (CUS), relating to the Client User Wallet. As core actors in ReSOLV, these systems represent roles that are similar to a client\server interaction, where both sides of the conversation are captured in the DFDs. The chapter concludes with a discussion of

the effectiveness of using DFDs to produce artefacts that describe the ReSOLV core processes.

5.2 Data Flow Diagrams

To further the research objectives of "designing a system, not making a prototype", an FD of the ReSOLV core actors is required to determine the processes that need to be built for the system. FD is the process where a higher level function is divided into identifiable sub-functions, corresponding to sub-tasks, and is a method for making part of the transition from function to structure in the design process (Al-fedaghi, 2016). FD can be contrasted with non-functional requirements, which specify overall characteristics such as scalability, usability, and reliability of a system (see Section 4.4.5 on page 141).

DFDs are system models, which show a functional perspective where each transformation represents a single function or process (Sommerville, 2010), and are traditionally used to model the system structure and interfaces as graphical representations. The purposes of DFDs are, to show data flowing through the system at an iteratively lower level, until all the processes that change data are identified, and to show where all the data is stored. DFDs differ from UML activity diagrams, which show transitions between activities and do not represent data flow. For the purpose of refining the ReSOLV process and functions, DFDs present a useful modelling approach to the data that flows between the various core actors within the ReSOLV ecosystem.

The structure of DFDs defines external entities that have system inputs and outputs to processes and data stores. DFDs are created in layers, which progressively drill into the functions where data is changed and stored (Hull et al., 2011). There are three principle layers for DFD modelling. The first layer is the Context Diagram, which represents a system that links only to external entities only (which be may other

systems). The second layer is the "Top level DFD" layer that outlines the processes and data that flows between external entities, data stores, and to other processes. The third layer (and any further layers) is used to further refine the data flows in each of the sub-processes that compromise the higher layer processes. In this manner, all data flows between entities, data stores and processes can be defined to the lowest atomic level, as required. Hence, as each layer is decomposed into its sub-processes, a greater level of detail is established, and a more accurate understanding of the function or process is also established. Figure 5.1 illustrates the DFD Layers, from the Context Diagram through to the sub-process layers.

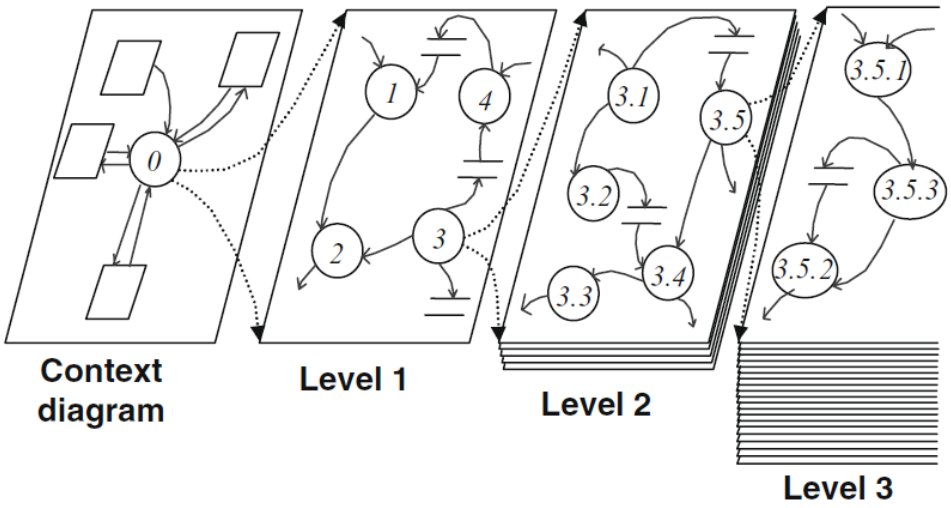


Figure 5.1: FD (Hull et al., 2011)

For the purpose of clarity in describing the data flows between the external entities, data stores, and processes, within the ReSOLV blockchain ecosystem, the following methodology has been applied to each DFD. The left side of the DFD is used for external entities, the right side of the DFD is used for data stores, and the centre of the DFD is used for the processes. The rules that are applied to the DFD to maintain consistency between the entities are:

- A process MUST change the data in some way
- Data can only come from a data store or an external entity
- Data cannot flow from one data store to another
- Data cannot flow from one external entity to another; it can only go through a process

There are two exceptions to these rules: (i) the Context Diagram only presents data from entities into a system, and hence data flows between external entities in a Context Diagram; and (ii) the cryptocurrency blockchain ecosystem introduces some complexities that necessitate the breaking of the data flow rules in some cases, requiring data to flow between external entities.

5.3 ReSOLV Functional Decomposition

5.3.1 Vendor Provenance System

This section presents the ReSOLV FD for the VPS, based on the ReSOLV RA. The VPS is fully defined through the Context Diagram, Top level DFD, and third level sub-process layers. Pseudocode (also known as mini-spec in DFD parlance) is included, which provides the context around the DFD process data flows. The last section also includes a data dictionary that explains the purpose for each data element and the processes that utilise it. In the blockchain ecosystem, some data elements are used in different ways, and may have different explanations.

The following figures present the DFDs for the VPS actor, taken from the ReSOLV RA. In the DFDs, each data flow represents data within a process, when a VPS actor for CUS is participating in the ReSOLV SLV method. The DFDs are, however, defining the required data for the VPS to function, whereas the ReSOLV RA simplifies the transaction process, without providing the detail necessary for developing a system.

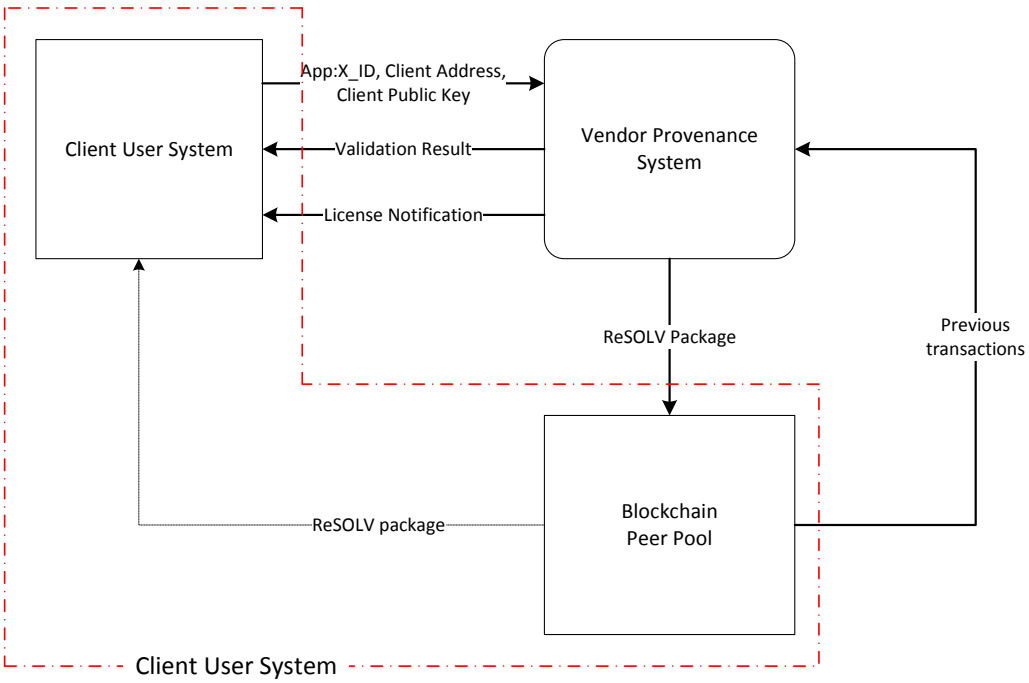


Figure 5.2: VPS – DFD Context Diagram

VPS Context Diagram

The VPS has inputs and outputs from two external entities: the CUS and the Blockchain Peer Pool (BPP) (Figure 5.2). The CUS sends the application related information to the VPS[1], and receives validation and license results. The VPS also has an input and an output from the BPP. In order to complete a licensing transaction, the VPS must validate the token it is holding, by reading the blockchain[2]. The output data from the VPS is the ReSOLV package that contains all the license information for the application. The final element to the Context Diagram is the ReSOLV package being sent to the CUS outside the VPS.

[1] Blockchain protocol transactions are inherently peer-to-peer. Architecturally, the information from the CUS is broadcast to the BPP, where the VPS will listen for information relating to it. For example, the application identifier can be the unique identifier that the VPS listens for.

[2] The VPS key store store has previously received a token from a different external entity, and separate DFD system.

VPS Top Level Diagram

The Top Level Diagram (Figure 5.3), is representative of all the processes in the underlying DFDs, with every entity, data store, and data flow mapped out in the Top Level Diagram. The Top Level Diagram shows that the VPS has been deconstructed into four sub-processes (third level processes), as listed below. All inputs and outputs to external entities shown in this DFD should reflect the Context Diagram.

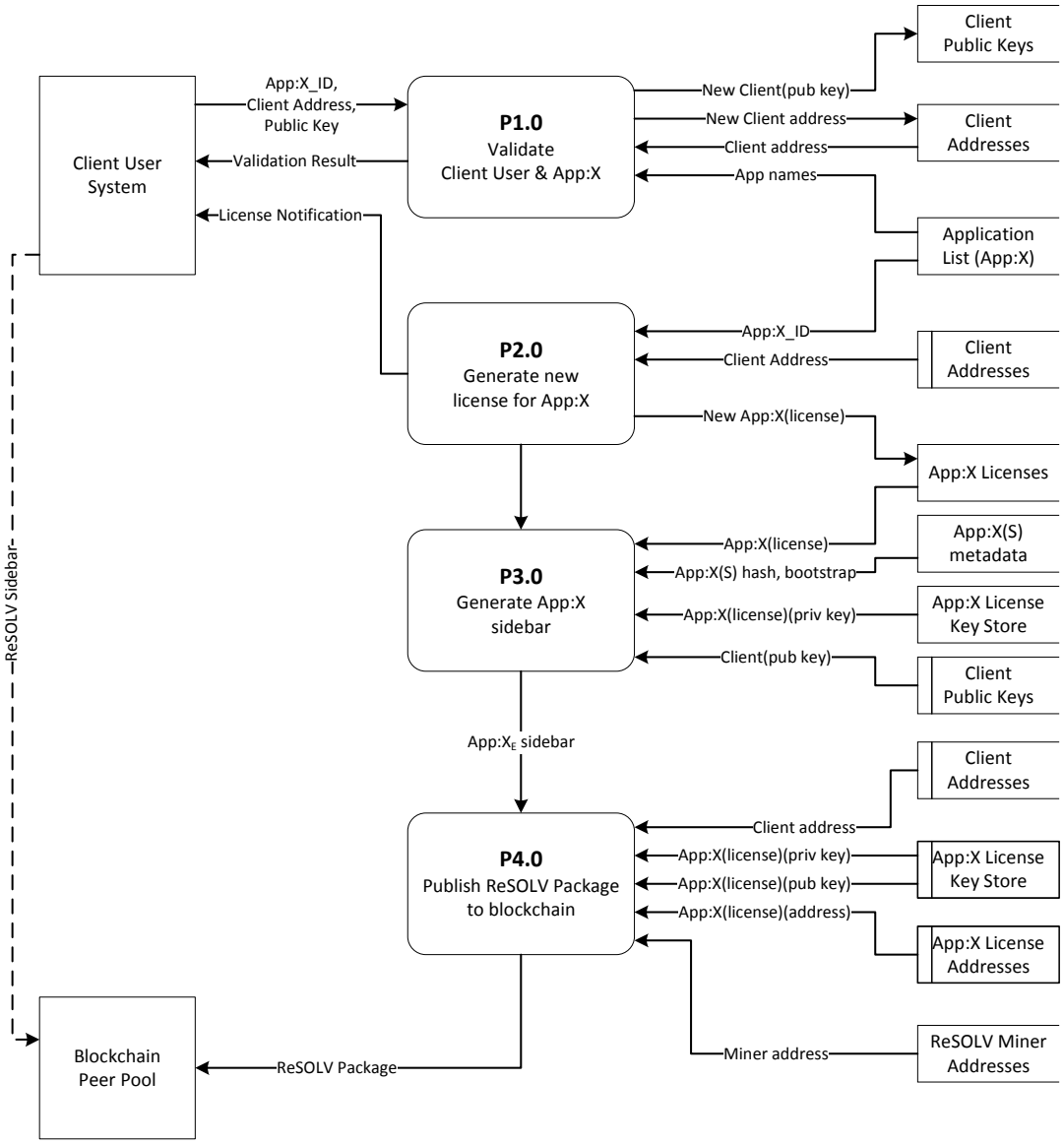


Figure 5.3: VPS – DFD Top Level Diagram

The four VPS DFD sub-processes, deconstructed and explained below, are:

- P1.0 Validate Client Address
- P2.0 Generate App:X License
- P3.0 Generate App:X Sidebar
- P4.0 Publish ReSOLV package to blockchain

P1.0 Validate Client Address

The Validate Client Address DFD (Figure 5.6 on page 166), deconstructs P1.0 into two sub-processes, P1.1 and P1.2. The DFD and pseudocode, together, show the purpose of this process is to validate the Client and store new Client details. The CUS also receives validation results from the VPS that ultimately can provide status information back to the human actor. This step is important as it creates the new client record if it does not already exist, and validates that the application is one that belongs to the Vendor. Where the new client record is being created, the premise is that the client has made payment for the software, and that a new license is to be issued to the end user (P2.0 below). This process satisfies RQ2.2, maintaining anonymity of the actors in a transaction.

P2.0 Generate App:X License

P2.0 is a single process that simply reflects generation of the license for the application (Figure 5.7 on page 167). The DFD and pseudocode together shows that the process is intended to utilise unique information from the Client to generate the license, called a tethered-license. In principle, this could provide a unique license that can only be executed from the Client User Wallet associated with the Client Address, rendering piracy through license copying impossible. This process is important because it is binding the end user identity with the application identity, and generating a new unique license that is entitled to run for an end user with a specific address. If the license is obtained through some man-in-the-middle type attack, the license is prevented from executing from any other address other than the one is bound to. However, the process

for license creation is beyond the scope of this research. This process supports satisfying RQ1.1 by ensuring that the validated user is assigned entitlements, and that the license cannot be used by any other actor.

P3.0 Generate App:X Sidebar

This DFD deconstructs the process for building the transaction Sidebar, which will be used by the User Wallet Agent when the application requests validation of a human actor (the user) into two separate processes, P3.1 and P3.2 (Figure 5.8 on page 168). The data contained within the Sidebar is critical to the process of preventing software piracy and protecting the application from at rest piracy. The DFD and pseudocode shows the data in the Sidebar includes the tethered-license, a hash of the application so that the application executable can be validated by the User Wallet Agent, and bootstrap code that will allow the application to run. Each individual data element in the Sidebar is then encrypted with the Client public key for confidentiality of the data on a public blockchain, and then a signature for the encrypted Sidebar (data only) is generated (using the private key associated with the license) and added to the Sidebar.

This DFD constructs the critical data for the ReSOLV method, and reflects the intention of the proposed ReSOLV model that was introduced in Section 4.3.2 on page 123. This process supports satisfying RQ2.1, ensuring that confidentiality of private data is maintained across the public blockchain, and supports RQ1.2 by demonstrating that platform-neutral data is placed on the blockchain.

P4.0 Publish ReSOLV package to blockchain

The purpose of P4.0 (Figure 5.9 on page 169), is to publish the ReSOLV package to the blockchain, so that the CUS can obtain the ReSOLV information required to validate the application and user entitlements. P4.0 is deconstructed into three separate processes, which first construct the blockchain protocol transaction inputs (P4.1). Then the outputs are constructed (P4.2) where the Client Address and license public key are added to the Sidebar. The license private key is used for signing the transaction, which

is needed when assigning the token to the Client User. This forms the ReSOLV package (a blockchain transaction), which is then broadcast to the BPP (P4.3) after receiving the reference ReSOLV mining DNS addresses.

P4.0 reflects the assembly of the necessary data to perform a transaction in a cryptocurrency or blockchain ecosystem, and transmit that transaction to the peer-to-peer mining network. This process supports RQ1.1, whereby the process actually "saves" the data to the blockchain as a transaction.

5.3.2 Client User System

In this section a partial ReSOLV FD for the CUS, based on the ReSOLV RA, is presented. The CUS is functionally established using the Context Diagram and Top Level DFDs. However, due to time constraints, DFDs that deconstruct the Top Level Diagram into the third level sub-process layers are not presented, and hence, no pseudocode is provided.

CUS Context Diagram (Figure 5.4 on the next page)

The CUS has inputs and outputs from three external entities: the Application System (AS), VPS and the BPP. The AS first initiates a request to the CUS to validate the human actor and receive the entitlements and data necessary to execute the application. This triggers the CUS to validate the user and the application identifier. If the application has been registered and licensed, the CUS will read the license Sidebar from the blockchain and send the necessary data to the AS. Where no license is stored, the CUS initiates a request license process to the VPS, to obtain a new license, as outlined in the previous section.

Client User System Top Level Diagram

The CUS Top Level Diagram (Figure 5.5 on page 163), shows that the CUS has been deconstructed into five sub-processes as described below. The descriptions are brief, reflecting the intended functions and possible sub-processes.

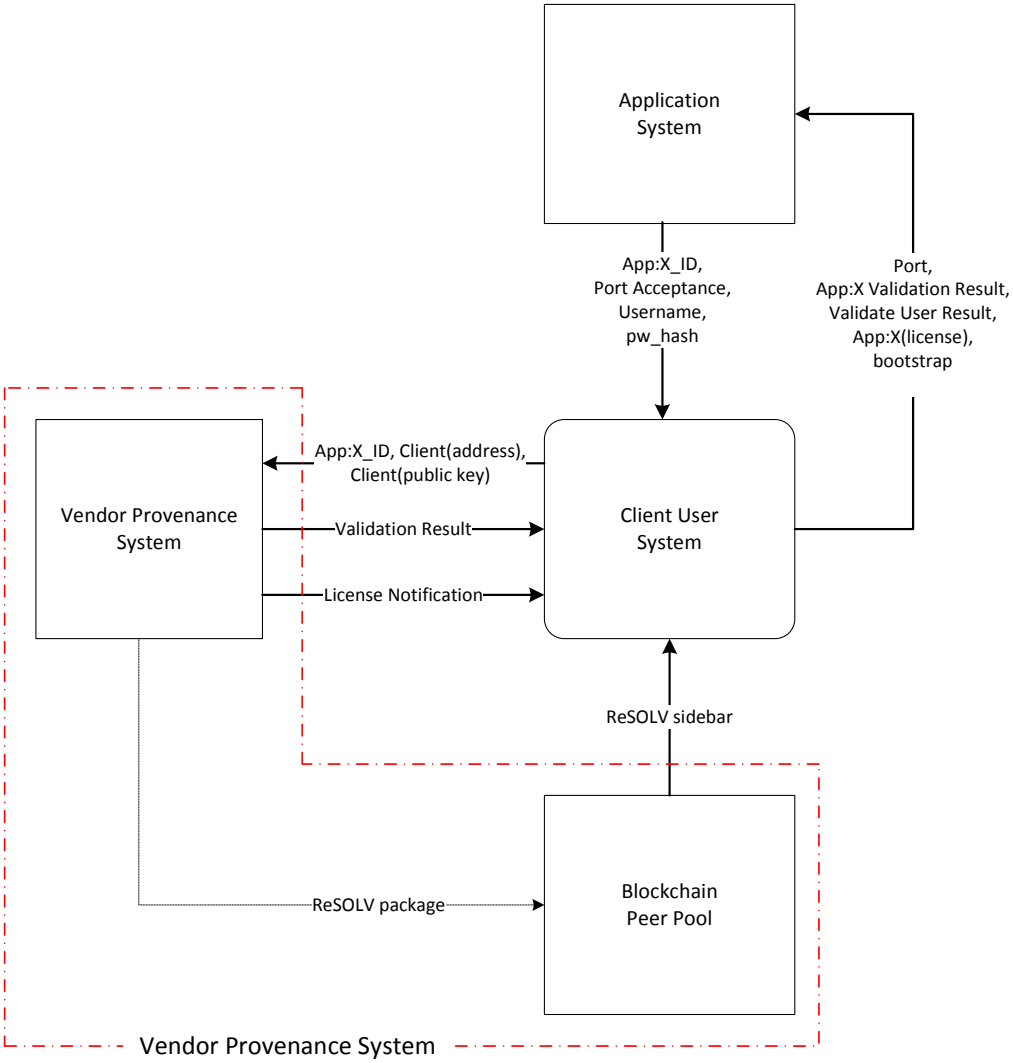


Figure 5.4: CUS – DFD Context Diagram

P1.0: Negotiate Protocols: when the AS communicates with the CUS, the two systems negotiate an ephemeral port unique to the application. This concept is similar to a Transmission Control Protocol / Internet Protocol (TCP/IP) three-way handshake. Because the port assignment is dynamic, the established port will be stored for use by all processes on a per session basis. This process is important because the CUS needs determine a unique port to maintain confidentiality of data over a separate channel, which supports satisfying RQ2.2.

P2.0: Validate Application: the CUS will first validate if a license for the application exists. If a license exists, the CUS will validate the application hash and a return result will be sent to the AS. This process is important because it determines if the application is valid and has maintained a state of integrity. If the AS application ID does not exist in the CUS, the CUS will initiate a process to request a new license from the Vendor (see P5.0 below). If the application ID exists, the AS will validate the integrity of the AS using the hash of the AS that is stored in the database (the blockchain). This process satisfies RQ1.3 though validation of the application by the CUS, using the stored hashes in the blockchain.

P3.0: Validate User: this process will likely have two sub-processes (P3.1 and P3.2) that reflect the clearly different functions. In P3.1, the User credentials from the AS are checked, and if no credentials exist a failed status result will be returned to the AS. Otherwise the credentials will be checked and passed/failed accordingly. P3.2 would facilitate the new user creation process to add a new credentials to the User database. This process is important for verification of the actor intending to use the application. Ultimately, this process will need to support multiple user authentication on a single device. It will also need to validate users against the blockchain, if no local Wallet for the specific user is present. This process supports RQ1.1 user validation requirements.

P4.0: Validate Entitlements: this process checks the license database for the application, and downloads the Sidebar from the BPP to validate or save the license entitlements. It sends the license and the application bootstrap back to the AS. This process is important because it determines any features within the application that the user is entitled too, and provides the bootstrap code that allows the AS to execute. This process supports RQ1.1 user validation requirements by validating the user against the ReSOLV blockchain, and also supports RQ 2.2, providing confidentiality of the actors.

P5.0: Request license from Vendor: where there is no license record on the license database, the CUS initiates a request to the VPS for a new license for the application. This process is important because it presents the user the opportunity to purchase a license from the Vendor through a disintermediated purchase process. This process also supports RQ1.1 user validation requirements.

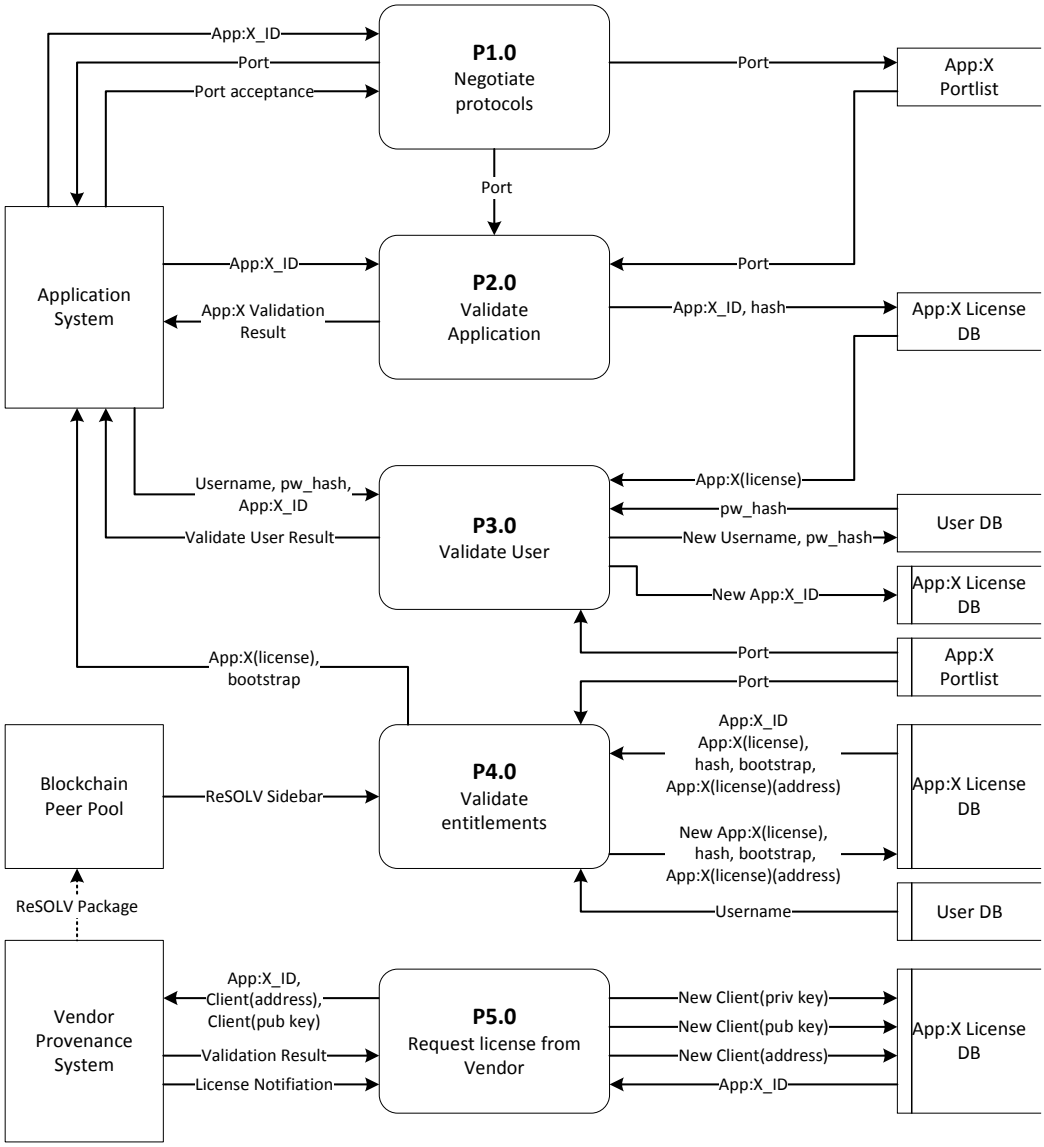


Figure 5.5: CUS – DFD Top Level Diagram

5.3.3 Data Dictionary

DFDs themselves do not provide any context around the purpose of the data and what the data flow relates to. Data Dictionaries are used to complement the DFDs and pseudocode to provide context for the data, to provide a brief explanation for the use of the data, and to describe the type of data being used (Crowder & Friess, 2013). A Data Dictionary for the VPS is introduced that provides the context around the data flows in the VPS process as shown in Table 5.4 on page 170. A Data Dictionary for the CUS is not presented as the sub-processes were not defined due to time limitations.

5.3.4 Analysis

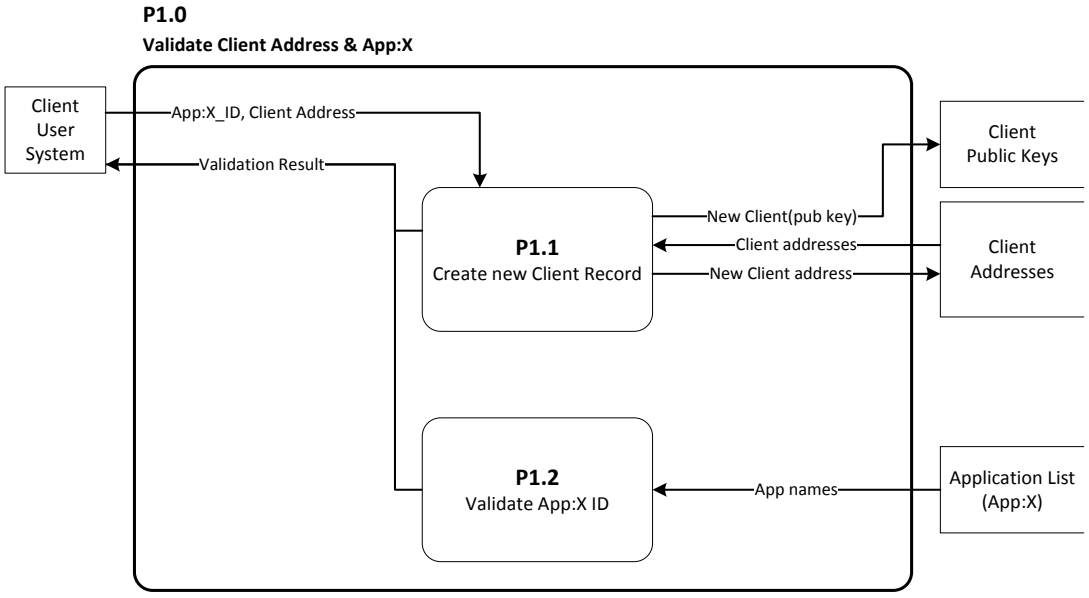
Throughout the FD process, DFDs are found to be effective, particularly where new ground is being covered, such as the process of creating a NBA, and very little reference information is available. Where Agile and SCRUM methodologies rely on existing references and expertise in creating particular types of systems, constructing the ReSOLV blockchain DFDs required a clear understanding of the blockchain functions to create the new artefact, and to understand the data flows between systems. DFDs force the understanding of the processes, and in this case the learnings were regarding the actual protocol level mechanisms of the Bitcoin protocol transaction. These learnings have been provided in Appendix A.2 on page 223.

A key interesting discovery, in respect to constructing DFDs, is that the blockchain protocol utilises a broadcast mechanism, where actors listen to broadcasts (which is facilitated by Miners), and don't interact with each other directly. This mechanism requires a modification to the DFD rules, as data needs to flow from sub-processes to other external entities (other systems) in the blockchain ecosystem.

5.4 Summary

This chapter discussed FD, and introduced DFDs as the method to determine the processes that need to be built for the ReSOLV system. DFDs and pseudocode are established for the VPS, and partially established for the CUS, to demonstrate how data flows between the two core technology actors. A Data Dictionary was also presented for the VPS, to provide the full context around each of the data elements.

This section concludes the artefact creation aspects of this research. The next chapter discusses the outcomes and findings during the artefact development process, and reflects on the Research Questions, hypotheses, and research methodology. It explores whether this research has succeeded in establishing that the ReSOLV method is viable for SLV.



Pseudocode

P1.1

```
If client address = null then

  Store Client address
  Store Client(pub key)
  Return Validation Result App:X_ID Received to Client

End If
```

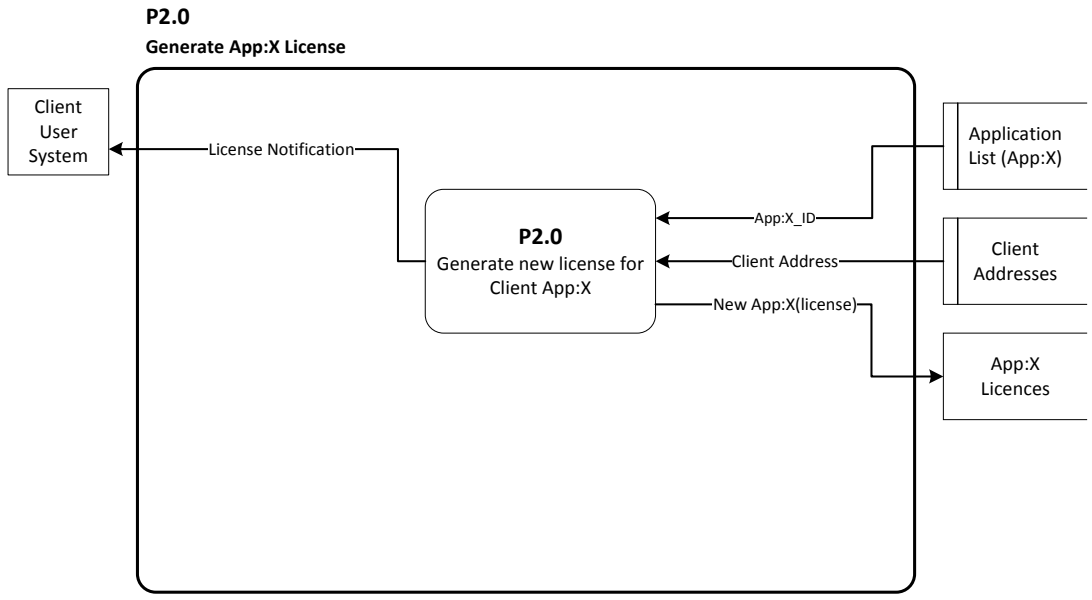
P1.2

```
If App:X_ID = null then

  Return Validation Result App:X_ID Unknown to Client

End if
```

Figure 5.6: VPS – DFD P1.0 Validate Client Address & App:X



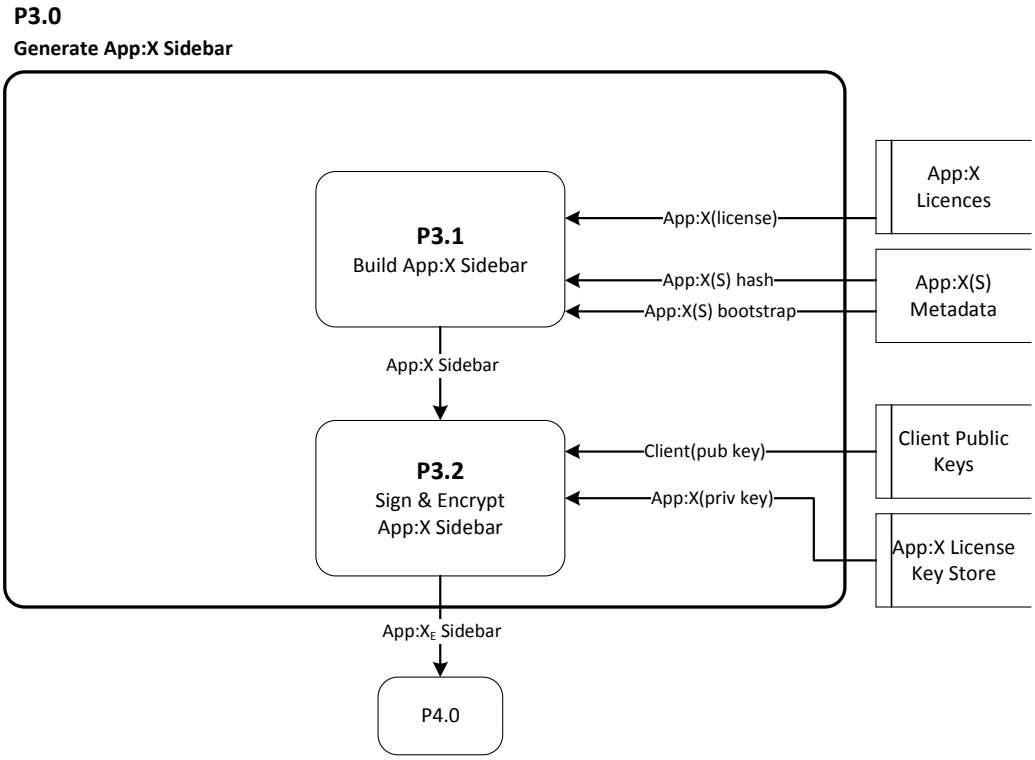
Pseudocode

P2.0

```
Use Client Address AND App:X_ID to generate license key for App:X_ID

Store App:X(license)as new license
```

Figure 5.7: VPS – DFD P2.0 Generate App:X License



Pseudocode

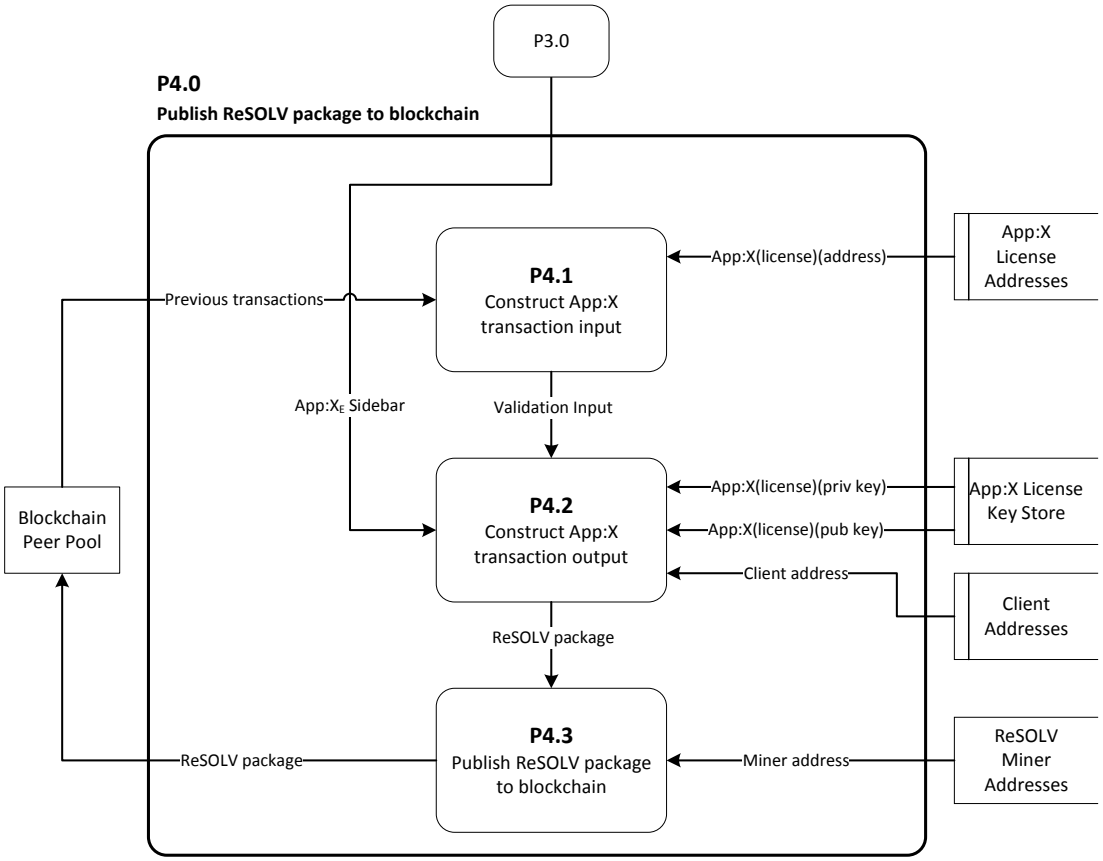
P3.1

```
Create Sidebar As String
Append App:X(license) to Sidebar
%% hash and bootstrap relate to the wrapped App:X(S) not App:X
Append App:X(S) hash to Sidebar
Append App:X(S) bootstrap to Sidebar
```

P3.2

```
Get Client(pub key)
Encrypt App:X(license) with Client(pub key)
Encrypt App:X(S)hash with Client(pub key)
Encrypt App:X(S)bootstrap with Client(pub key)
Get App:X(license)(priv key)
Hash Sidebar with Client(pub key)
Digitally sign Sidebar with App:X(license)(priv key)
Append Digital Signature to Sidebar
```

Figure 5.8: VPS – DFD 3.0 Generate App:X Sidebar



Pseudocode

```
P4.1  Transaction As String
      Append App:X(license)(address) to Transaction  %% FROM address
      Get input transactions from blockchain
      Create hash digests of input transactions
P4.2  Append hash digests to Transaction             %% Current validation
      Append Client address to Transaction           %% TO address
      Append App:X_E Sidebar to Transaction          %% License data
      Append App:X(license)(pub key)                 %% Future validation
      Create ReSOLV package Signature using App:X(license)(priv key)
      Append ReSOLV package Signature to Transaction
      %% ReSOLV package now created
P4.3  Get Miner Address to resolve blockchain Peer Pool
      Push ReSOLV package to blockchain Peer Pool
      %% Miners now need to validate transaction and add to blockchain
```

Figure 5.9: VPS – DFD P4.0 Publish ReSOLV Package to Blockchain

Table 5.1: ReSOLV Data Dictionary

Variable_Name	Variable_Description	Used_By	Type
Client_Public_Key	Stores the Client public key that will be used to encrypt the license data in the sidebar	**P1.1:** Used to store new client public keys; required for encrypting client data **P3.2:** Used to encrypt client data in the Sidebar that will be placed on the blockchain	String: ECDSA: Public_Key
Client_Addresses	Stores the requesting client address so that the correct address for the license is placed on the blockchain	**P1.1:** Used to validate the client address (Public Key and Client Address should match, both can only be created with the Client Private Key) and add a New Client address **P2.0:** Used to generate a Client address tethered-license App:X(license) **P4.2:** Used in the construction of the transaction output. It is the destination address (the Client)for the ReSOLV package	String: ECDSA: Public_Key
App_X_ID	The unique ID for each application released by the software vendor	**P1.2:** Used to validate that the App:X ID exists. In the future, this may also indicate any license specific terms related to the license **P2.0:** used in the creation of App:X(license)	String: Token_Address
Validation Result	Stores the result status for the User and App:X_ID validation function	**P1.1:** Used to return status of valid or invalid for the Client validation **P1.2:** Used to return the status of valid or invalid for the App:X_ID validation	Integer

Table 5.1: ReSOLV Data Dictionary

Variable_Name	Variable_Description	Used_By	Type
License Notification	Stores the result status for the license generation function	**P2.0:** Used to return status of complete or failed for the license generation	Integer
App Names	Stores the result of a query for App:X_ID	**P1.2:** Used to determine if App:X_ID is valid	String
App:X(license)	Stores license for App:X_ID	**P2.0:** Used to record the new license in the license database **P3.1:** Used for the construction of the App:X Sidebar	String
App:X(S)hash	Stores the App:X(S) hash	**P3.1:** Used for the construction of the App:X Sidebar	String
App:X(S)bootstrap	Stores the App:X(S) bootstrap	**P3.1:** Used for the construction of the App:X Sidebar	String
App:X(license)(priv key)	Stores the private key of App:X(license)	**P3.2:** Used to digitally sign the App:X Sidebar **P4.2:** Used to digitally sign the ReSOLV transaction	String: ECDSA: Private_Key
App:X(license)(pub key)	Stores the public key of App:X(license)	**P4.2:** Appended to the ReSOLV transaction and used by anyone to validate the transaction in the future and provide authorisation “spend” the token	String: ECDSA: Private_Key
App:X(license)(address)	Stores the address of App:X(license)	**P4.2:** This is the source or current address of the token being used in this transaction	String: ECDSA: Public_Key

Table 5.1: ReSOLV Data Dictionary

Variable_Name	Variable_Description	Used_By	Type
Miner Address	Stores an IP address	**P4.3:** This is the IP address of the DNS servers used to resolve the core Miners to establish who to connect to in the Blockchain Peer Pool	String:
ReSOLV Package	Stores the fully constructed ReSOLV package data	**P4.3:** This is full transaction that will be processed by the Miners in the Blockchain Peer Pool	String:
App:Xϵ Sidebar	Stores the encrypted App:X Sidebar	**P3.2:** This used to hold the encrypted Sidebar that will be appended to the ReSOLV transaction	String:

Chapter 6

Discussion

6.1 Introduction

In the last chapter, FD for the ReSOLV method was discussed, and Data Flow Diagrams were introduced, as the method for determining the processes required in the ReSOLV RA. This resulted in the creation of Data Flow Diagrams for the core VPS and CUS, with pseudocode completed to provide context for the data flows between these two core technology actors. Together, these artefacts demonstrated how the hypothetical SLV method would function.

In this chapter, the findings from the RE and FD chapters are presented and discussed in respect to the Research Questions established in Section 2.8.1 on page 91. Each Research Question is examined, to determine the level of success resulting from the research outputs, and the overall success of the research outputs is evaluated, with outcomes confirming the hypotheses established in Section 2.8.2 on page 92. Lastly, reflections are provided, to evaluate the suitability of the DSR approach to test the Research Questions, and to consider improvements that could be made for future research.

6.2 Research Motivation

For the purpose of grounding this chapter, a review of the motivations and drivers for this research into solving a global software piracy problem is required.

The review of literature in Chapter 2 on page 27 explored the scope and depth of software piracy, and confirmed the complexity of addressing the issues surrounding it. Several models that improved the understanding of piracy issues were presented, forming a clearer picture of how piracy occurs, and the relationships and methods used to obtain pirated software. Together, these define the software piracy attack surface, and are summarised as follows:

Establishing the scope of software piracy in respect to global impact, and expansion of platform piracy issues into the mobile computing world and gaming platforms (as discussed in Section 2.2 on page 28).

Identifying, defining and relating software piracy types and actors involved in software piracy, concluding by introducing a taxonomy of software piracy types (and roles (as shown in Table 2.1 on page 51).

Determining the methods used to protect software from piracy, and how threat actors defeat piracy protection mechanisms(Section 2.2 on page 56) and overcome prevention methods (as discussed in Section 2.4.4 on page 60).

Relating how the threat actors interact and engage in the software piracy process, either deliberately or unwittingly, through the software piracy model (as shown in Figure 2.7 on page 59.

Identifying the points where software piracy can occur as shown by the Software Vulnerability Piracy Lifecycle (Figure 2.8 on page 64).

Identifying related software piracy issues such as enforcement, jurisdictional boundaries (see Section 2.4.7 on page 65), and how pirated software is a well known vector for malware (see Section 2.4.8 on page 67).

The attack surface shows that the path to software piracy prevention, and protection of software creator copyright, is challenging. Piracy is still rife across all ecosystems other than Apple IOS, and even cloud computing business models suffer considerable piracy due to credential sharing. The aggregated opportunity cost of software piracy across the various platforms is estimated to be USD $132 billion annually, excluding economic costs from mobile and cloud platforms – because there is little or no peer reviewed research available for these platforms. Hence, there is strong motivation to solve this problem.

A blockchain-based SLV method for software piracy prevention and provenance needs to:

Be Global: it should be available online so that software creators have knowledge of use of their software.

Support Multiple Platforms: it needs to be a single system that is supported cross-platform, so that it is straightforward for software creators building on any platform.

Be Cost-effective: this is mainly a product of being homogeneous and simple,but also reflects a requirement for minimal infrastructure.

Place license authorisation in the hands of the software creator: the software creator should have control over license issuance.

Protect software at rest: to prevent reverse engineering and cracking of software by threat actors or malware.

Uniquely bind user identity to software entitlements: so that only licenses issued legitimately, to unique users, are valid.

Disincentivise credential sharing: by multiple parties using the same software.

Provide end user anonymity and data confidentiality: by reducing the attack surface across all states of the software piracy vulnerability lifecycle.

There is *one final requirement* for this method, which derives from the literature review. A key learning from that review was that the actor role is the fundamental lowest common denominator in software piracy (as shown in the software piracy process in Figure 2.7 on page 59), and that user oriented entitlements ensure that only a specific user has software entitlements, as shown in the SPVL in 2.8 on page 64. The requirement then, is for a strong mechanism to be established between the **User** and the **Software Entitlement**, where end users cannot, or are very unlikely to, share credentials.

Further to this, the review of literature shows that cryptocurrency and blockchain technologies[1] are a feasible technology to meet the requirements to protect software across the attack surface and, therefore to overcome the piracy issues. The findings show that blockchains are global, distributed, cost-effective, and customisable. Applications can be written to leverage existing cryptocurrencies, or can be designed or created as a new ecosystem dedicated to a specific use-case, called NBAs. Based on existing proof-of-concepts and industry developments that demonstrate new use-cases for blockchain technology, it was determined that SLV could feasibly be constructed using the technology. Building on this, the literature review discovered that an existing proof-of-concept, called the MBM, had proven that a cryptocurrency could be used to convey license entitlement in a basic sense.

In summary, a motivation with clearly stated problems has been defined, and a potential method to address the problems, using blockchain technologies (specifically a NBA), has been established. With the motivation established, the following sections discuss the findings and outcomes of this research.

[1] It should be noted that since the inception of this research, a new term has evolved within industry to describe systems that utilise blockchain-like technology, called "Distributed Ledger Technologies" or DLTs. DLTs use new mechanisms such as hashgraph (Baird, 2016), and can be quite different from the blockchain, but solve inherent problems that blockchains can present for certain use-cases. For consistency of this research, blockchain technologies will continue to be used.

6.3 ReSOLV: a Native Blockchain Application

6.3.1 Findings

The purpose of RQ1, as established in Section 2.8.2 on page 92, was to determine if the Native Blockchain Application (defined in Section 2.6.2 on page 80) can be applied to meet the functional requirements of a distributed SLV method. A review of methodologies, in Chapter 3, determined that DSR was the most suitable research methodology given that cryptocurrency software engineering is highly iterative as new ideas are designed, tested and evaluated.

RE in Chapter 4 on page 113, examined the MBM license provenance proof-of-concept, and its application to SLV (Section 2.7 on page 85). The findings demonstrated that the MBM was applicable to aspects of SLV. However, MBM is too simplistic in that it fails to take account of the complex nature of software licensing and distribution, as identified by proposed alternative SLV methods, such as Pirax, DRM Framework and Tagged Transaction Protocol (Section 2.7 on page 85). Furthermore, MBM provides no protection against most of the forms of software piracy identified in Section 2.4.5 on page 61. Examination of the software piracy process (Section 2.4 on page 52) also demonstrates that there are many methods that circumvent the MBM protection mechanism. These include techniques such as removing the code that validates the entitlement on the blockchain, or sharing of the cryptocurrency private keys used by the software licensing method.

Building on this, the ReSOLV Model was proposed to overcome the limitations of the MBM by including license data stored on the immutable blockchain in encrypted form, using well established public-key cryptography methods. The proposed license data stored on the blockchain includes license key, software hash, bootstrap loader, and a digital signature field, providing a global method for software license distribution.

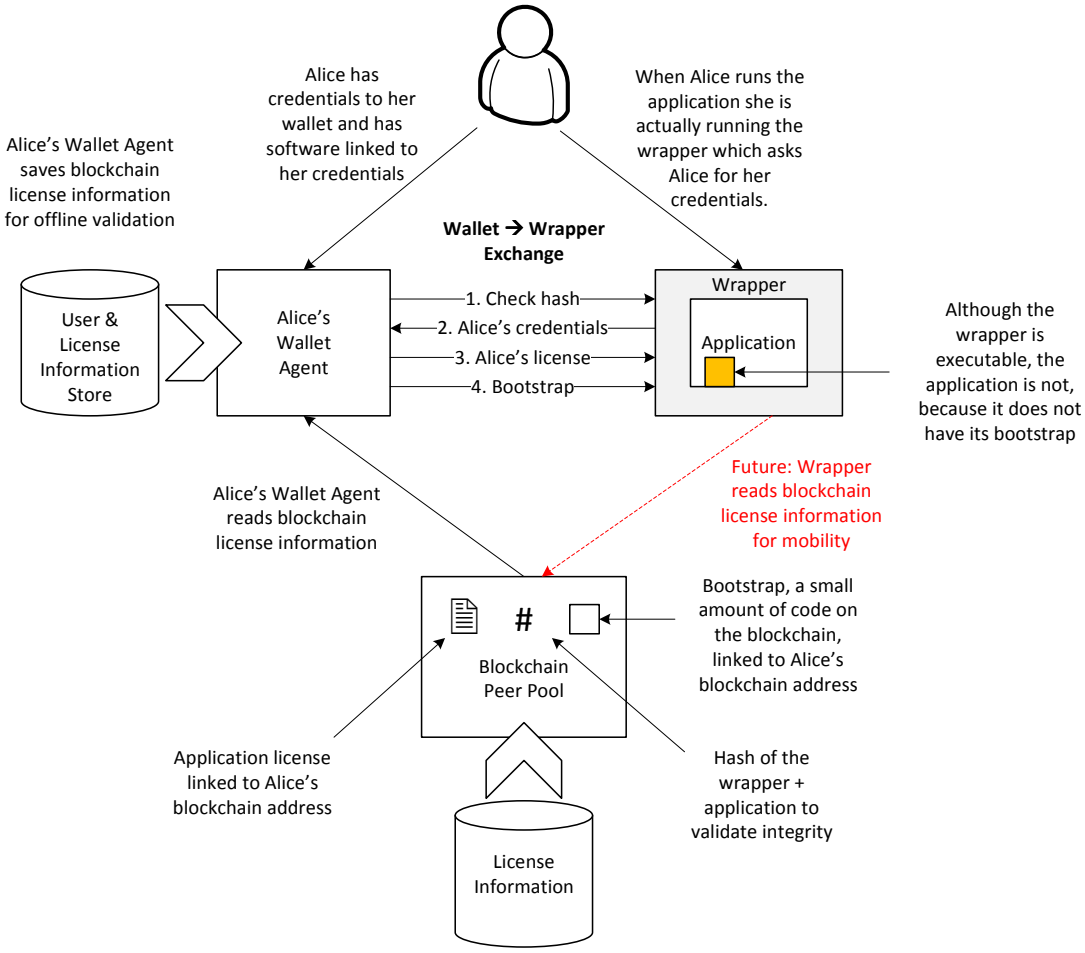


Figure 6.1: Simplified ReSOLV – Client-side

ReSOLV Client-side Research Outcomes

From a post-requirements specification and design perspective, Figure 6.1 illustrates the ReSOLV SLV concept from the client-side. This simplified diagram is taken from the ReSOLV RA and presents the core client-side innovation – a ReSOLV wallet agent and an application that has a wrapper applied, which is executed ahead of the application to validate the user. Using Alice as the user, Figure 6.1 shows that Alice has a Wallet Agent that reads license information from the BPP, which it also stores in its local database (User and License Information Store). The database also holds user credentials and private keys related to the licenses, an essential requirement for the blockchain

method to function. The application is "wrapped" with a ReSOLV executable, which first communicates with the Wallet Agent to validate the application's integrity using the stored hash of the application. It then requires Alice to enter her credentials to authenticate to her Wallet Agent, which, if Alice's credentials are correct, matches her entitlements and provides the application wrapper with the license and bootstrap.

Some key considerations from the discussion include the following:

- For the purpose of this research, the ReSOLV wrapper itself has not been defined. From a proof-of-concept perspective, it may just be a separate application that subsequently launches the application. In practice, the wrapper may indeed wrap the application, or it may utilise libraries that are included during software development.
- Using the Wallet Agent as an independent active agent, it provides the validation of both the application and the user. The hash check of the application is fundamentally powerful because it prevents application execution in the event the application has been modified through malicious cracking or malware behaviours. The hash itself cannot be modified because the hash is digitally signed by the vendor and held on the immutable blockchain, so an attack on the Wallet Agent will not result in a compromised license.
- As demonstrated in Section 4.3.2 on page 123, the ReSOLV model supports application changes such as patching or upgrades. The application integrity is protected throughout its lifecycle, and vendors can manage re-licensing applications throughout the software development lifecycle. However, it should be noted that this research has only explored the process of upgrading a single executable. As applications may have many libraries and executables, consideration should be given to what needs to be included as part of the ReSOLV model.
- The FD of the ReSOLV Vendor Provenance function resulted in the proposition

of the license key being generated, utilising the Client license address to provide an entirely unique license that may only be run from the User Wallet address. This prevents any utilisation of the license key if the threat actor has obtained the user's private key and has successfully hacked the license from the Wallet Agent database. In this instance, the license will not function in any other Wallet Agent. Furthermore, as the license key is digitally signed by the vendor, a threat actor cannot sign a license with his own key. Hence, this method protects the license integrity and ensures the license used is the one issued by the vendor.

High Level Architecture

The HLA provided more of an enterprise perspective of ReSOLV and what the ecosystem may look like when integrated into third-party services for authorisation and support of enterprise directory environments, such as Microsoft's Active Directory. There will be future challenges, managing license entitlements for employees who will eventually depart the business that hired them. There will need to be a method for licenses to expire to allow for fluid situations, such as a dynamic workplace. The ReSOLV mechanism already supports this through the software update mechanism described in Section 4.3.2 on page 123, but would need to be refined.

Further challenges are identified in the HLA, primarily concerning the non-functional requirements, where scalability and robustness of the ReSOLV ecosystem must be considered as part of the RE process.

Vendor Provenance Agent

The Vendor Provenance Agent presented an interesting challenge during the requirements specification and FD. It was quickly, apparent during requirements specification, that where detailed requirements were needed, they would need to be limited to functional requirements. The non-functional requirements for an NBA are considerable, and require research in their own right. In addition, the ReSOLV HLA was far too

large from a functional requirements perspective. This resulted in the development of the ReSOLV RA, to provide focus, and identify the core actors and functions of the ReSOLV model.

The ReSOLV RA also lacked detail for the DFDs in the FD (see Chapter 5 on page 152). This required some careful investigation and consideration of the underlying Bitcoin protocols used by the ecosystem actors, necessitating a re-architecture of the Wallet Agents in the ReSOLV RA.

A useful outcome of the DFD process was to identify that Bitcoin as a cryptocurrency was not suited to the ReSOLV model. Bitcoin supports up to 80 bytes of data in the Bitcoin protocol. However, ReSOLV utilises 76 bytes across license key, hash digest, and signature, leaving no room for the bootstrap. Ethereum supports larger data amounts, and could feasibly be used for ReSOLV except that every transaction incurs an individual gas charge, which would render ReSOLV non-viable from a cost perspective.

The following sections respond directly to the Research Questions proposed for this research.

6.3.2 RQ1.1

How can the Native Blockchain Application be utilised to protect software copyright by validating user entitlements?

Through the RE and FD design process, it was established that NBA would be the most suited for developing the ReSOLV model. The blockchain based method provides the core mechanisms for storage and distribution of the software licenses and data used to validate the application. This method eliminates man-in-the-middle attacks by using digital signatures from the vendor. The key points supporting this finding are:

- Licenses are protected from any copying process, as they are uniquely keyed to the Wallet Agent address and will not function anywhere else.

- Applications will not execute if any unauthorised modification of the application is undertaken.
- Applications will not execute without the bootstrap. If an attack attempts to capture the bootstrap in memory and insert it into the application, the application hash will not match, and the application will not execute in any case.
- Although a user can disclose their credentials and private keys to the Wallet Agent, the user is disincentivised to do so, because that would expose the license details for all their license entitlements. This is similar to giving someone your EFTPOS card and PIN to keep, so they can withdrawn money once. It would be a high risk proposition.

On reviewing suitability of existing cryptocurrencies, such as Bitcoin and Ethereum, for SLV, it was found that these could provide the required technical functionality. However, the research contends that non-functional considerations, such as cost, and ecosystem considerations such as scaling, will create a persuasive weighting towards implementing ReSOLV as an NBA.

6.3.3 RQ1.2

How can the Native Blockchain Application validate user entitlements across multiple platforms?

Given that the requirements specification is platform neutral, it is considered practical to state that the ReSOLV agents can be written for multiple platforms, such as Microsoft Windows and Android. This is demonstrated through the ReSOLV RA, the DFDs, and associated pseudocode, which demonstrate there is no bias for operating system or platform.

The ReSOLV method utilises the blockchain for license and application protection purposes, without a requirement for knowledge of the platform. This is shown by:

- The license and hash mechanisms are not application data, and hence are defined as platform neutral data.
- The bootstrap must be operating system specific, but it is related to the application identity (App_ID), as shown by the DFDs for the Vendor Provenance Agent. Although the FD for the Software Encoding Service is not included in the scope of this research, it will extract a bootstrap from the application code and store it in the Provenance Database (as shown in Figure 4.7 on page 137). This will subsequently be used in the ReSOLV Sidebar, as shown in the VPS Top Level Diagram (Figure 5.3 on page 157).
- Blockchain client applications support multiple platforms, with cryptocurrency wallets being developed for multiple operating systems, including Windows, Apple MacOS, Apple IOS and Android.

Finally, to achieve validation of user entitlements across multiple platforms, each supported platform will require a User Wallet Agent and Application wrapper, developed for that platform.

Further Considerations

Other considerations, in respect to multiple platforms, became apparent during this research. Although this instantiation of design shows the ReSOLV method supports multiple platforms, the practical aspect of User Wallet mobility needs to be considered. The User Wallet Agent contains the private keys that relate to each application license stored in the User Wallet database. For a user to be fully mobile, the User Wallet must also be mobile, to support the user authenticating from multiple devices. In addition, User Wallet security must be considered, as the User Wallet holds all the keys to the "crown jewels" and cannot be recreated if lost.

In respect to cryptocurrencies, this problem has been partially addressed by wallet

developers, such as those who developed the Jaxx wallet[2] or Exodus wallet[3]. Known as "hot wallets", these applications have the capability to store private keys locally, and provides a 12-word backup phrase that can be entered into a wallet on another device, to allow a synchronisation of private keys. Whilst this is a useful backup mechanism, which provides a modicum of license mobility[4], it does not solve the user authentication problem if the User Wallet is not present.

Some approaches for addressing these problems may include: (i) using a portable device to store the license private keys (for example, a special hardware wallet, such as the Trezor[5]); (ii) developing the Application wrapper to read license information directly from the blockchain; and (iii) using cloud-based authentication services for the users, as shown in the HLA (Figure 4.6 on page 133).

6.3.4 RQ1.3

How can the Native Blockchain Application be used to identify maliciously modified software?

For clarity, the context for this RQ is in respect to the ReSOLV SLV method. The findings that relate to this question show that the ReSOLV client can detect unauthorised changes to a file with certainty, but it cannot ascertain if the unauthorised changes were malicious (it simply detects that the hashes do not match). ReSOLV has two processes supporting identification of unauthorised, modified software:

[2] https://jaxx.io/
[3] https://www.exodus.io/
[4] From a user experience perspective, it is onerous to enter in a 12-word backup phrase. Furthermore, wallet developers are increasing the backup phrase to 24 words for improved security, as a result of a client-side hack on the Jaxx wallet. http://bitcoinchaser.com/jaxx-hack-400k-stolen.
[5] https://trezor.io/

- The software upgrade mechanism, as described in the ReSOLV Model (Section 4.3.2 on page 123), ensures that every hash of every authorised application and its subsequent patches are recorded on the blockchain.
- The ReSOLV User Wallet Agent acts as an independent authority, checking the hash of the application, based on the digitally signed hash on the blockchain (or in its database).

Further Considerations

From a practical perspective, applications may comprise hundreds or thousands of files. The ReSOLV method in this initial design is predicated on preventing the main application executable file from being tampered with, rather than detecting modifications to files within the entire file structure. It would be impractical to have a hash of every executable, library, or file that applications may require during installation.

However, this problem could well be solved by borrowing a cryptographic mechanism from the Bitcoin design. In each block of transactions, Merkle Trees are used to consolidate transaction hashes into a single hash to provide an integrity statement for all the transactions in the block (Nakamoto, 2008). Applying this method to a ReSOLV application, the vendor could provide a single hash for all the files required for the application, thereby allowing the ReSOLV User Wallet Agent to detect unauthorised modifications to any files. This would not be able to provide information about which file had been modified, however; this would need to be completed through some other installation integrity checking process.

6.4 Cryptocurrency-neutral Software License Validation

6.4.1 Findings

The purpose of RQ2, as established in Section 2.8.2 on page 92, was to determine which cryptocurrency provides the most suitable base for the SLV use-case.

The intention for these research questions was to investigate the non-functional requirements related to the ReSOLV method. It also was intended to create an enquiry into the characteristics of existing cryptocurrency ecosystems, to determine their suitability for the ReSOLV method. Non-functional requirements are researched, briefly discussed, and outlined, in Chapter 4 on page 113 for completeness. However, the scope of non-functional requirement research was subsequently deemed too large, for the purposes of this research in respect to time constraints. Hence, although valuable insights would be provided, research of cryptocurrencies and their characteristics was not pursued, due to the depth required for each non-functional requirement.

Having said this, the ReSOLV RA (Figure 4.7 on page 137) and the Vendor Provence System High Level Diagram (Figure 5.3 on page 157) demonstrated that data confidentiality and user privacy requirements are independent of the base cryptocurrency used. Hence, the ReSOLV model can be determined to be cryptocurrency neutral in respect to data confidentiality and privacy requirements.

The following sections respond to their respective research questions pertaining to how data confidentiality and user privacy are maintained independent, of the underlying cryptocurrency or blockchain.

6.4.2 RQ2.1

What cryptocurrency characteristics are required to achieve data confidentiality in a blockchain-based SLV method?

The research determined that data confidentiality is a functional requirement. Any data that is placed on the blockchain is expected to be encrypted, as the blockchain is a public ledger, which any actor can read. Hence, to maintain data confidentiality, encryption is required. This is supported by the following findings:

- The most common form of encryption is asymmetric private-public key, which cryptocurrency and blockchains are built on and is explained in Section 4.4.6 on page 144.
- Utilisation of existing private/public key pairs for encryption of data on the blockchain provides effective encryption of data on the blockchain, and therefore, effective confidentiality.
- Only the private key holder can decrypt the data on the blockchain. In the ReSOLV method, this is the User Wallet Agent.

In response to RQ2.1, this research contends that data confidentiality is independent of blockchain characteristics and is a functional requirement defining data encryption requirements. Data confidentiality is confirmed using standard private-public encryption methods.

6.4.3 RQ2.2

What cryptocurrency characteristics are required to achieve actor privacy in a blockchain-based SLV method?

Anonymity on cryptocurrencies and blockchain is an inherent characteristic, due to the address creation mechanisms using ECDSA private-public key cryptographic methods. This requirement is determined to be a functional requirement, which is considered to be cryptocurrency neutral. However, for Bitcoin it has been possible to identify actors, through identification of address endpoints in the real world. Anti-money laundering and

fraud investigators have concentrated on this, in an effort to prevent Bitcoin being used for payment in illegal transactions. However, actor privacy is considered a fundamental requirement in the cryptocurrency world, with new architectures introduced to solve the quasi-anonymity problem of Bitcoin. Cryptocurrencies such as Dash[6], Zcash[7] and Monero[8] are all providing fully anonymous transaction capabilities.

Confirmation of actor privacy in the ReSOLV model is supported by the following findings and outcomes of the research:

- Anonymity is achieved through the cryptocurrency address creation mechanisms, which generate strongly unique addresses with a negligible chance of collision (the Bitcoin address space is 2^{160} addresses). Addresses are then used as the identifiers for accounts in the cryptocurrency ecosystem.
- Each license has a source unique address from the Vendor. This means that every software item purchased from this Vendor has a unique address, and transactions to other clients cannot be identified as being initiated from the Vendor.
- Each User Wallet Agent creates a new license address when requesting a license from the Vendor Provenance Agent, as shown in Figure 5.3. This ensures that the Clients cannot be identified by parties reading the blockchain.
- Data confidentiality through private-public encryption maintains the confidentiality of any Client information contained within the transaction.

Further Considerations

As a counterpoint to the anonymity that is expected from the cryptocurrency ecosystems, ReSOLV software Vendors will want to know their customer base. Customer identity is relevant to provenance and pricing, whilst "knowing your customer" will be valuable for the Vendor to help them define and add features to their products. Financially speaking,

[6] https://dash.org
[7] https://z.cash
[8] https://getmonero.org/

the value of products, and intangibles such as goodwill, requires that customer purchase history is easily available to the Vendor (but unavailable to everyone else).

Furthermore, from the Client perspective, there may be other Client information that needs to be announced to the BPP, which requires consideration of privacy. Most examples will be provenance or pricing related. For example: (i) a student may have a cheaper price; (ii) a corporate may have a special agreement with a Vendor; (iii) a supply chain re-marketer receives a commission for orders through their business; or (iv) a Client may need to update their business details. Hence, data identifying the User may need to be included in the license request process, and will need to be encrypted for confidentiality, using the Vendor's public key for the license.

6.5 Reflections on Design Science Research

The objective of this research was to design a system and create artefacts that supported the testing of the Research Questions. The DSR methodology was selected, based on the likely requirement for iterative design through the RE and design process.

Key Findings

The requirements specification was highly iterative, particularly for development of the ReSOLV HLA (Figure 4.6 on page 133) and the ReSOLV RA (Figure 4.7 on page 137). Development of the GWT user stories provided context to the RA, and also identified further refinements required to achieve the SLV functions. Once the requirements specification was completed, it transpired that there was insufficient detail to be able to describe the data flows involved in the transactions between actors in the ReSOLV model. In addition, the user stories lacked clarity as to how the ReSOLV architecture supported the Research Questions, and how the actual transaction occurred within the blockchain protocol.

To remedy this lack of clarity, a deeper investigation into the blockchain protocols was required. A FD was undertaken to define the processes and data flows for two key actors: (i) the Vendor Provenance Agent; and (ii) the User Wallet Agent. DFDs were completed for these, to determine the processes and data transformations that occur, within the Agents and between the Agents, in the ReSOLV ecosystem, for an end-to-end SLV transaction. This required further research into the Bitcoin transaction to a protocol level[9], to learn how transactions are crafted, byte by byte (similar to how TCP/IP packets are constructed). These findings necessitated a further modifications to the ReSOLV RA, with changes flowing through to the user stories.

For the situation where the system is an unknown quantity during the RE design phase, completing the DFDss produced an effective outcome that increased clarity of the design requirements. The UML sequence diagrams, shown in the ReSOLV model in Section 4.3.2 on page 123, are helpful in articulating the process state, but were insufficient to describe the requirements from a specification perspective. Other considerations included whether AOSE (Section 3.5.3 on page 110) would be an effective design methodology for the cryptocurrency and blockchain ecosystem.

DSR is found to be an effective research methodology for testing the hypotheses and Research Questions, and creation of the supporting artefacts.

Improvements

Although using a DSR methodology has been effective in achieving the research outcomes, there are several improvements that have been identified and provide useful discussion points.

- The Agile methodology is most commonly used for software development, from the perspective of software engineering, a SCRUM type approach is likely to find development of a novel artefact challenging, unless there is reasonable prior

[9] A summary of the Bitcoin transaction findings can be found in Appendix A.2 on page 223.

experience in the product that is being created. The observations made during the RE process demonstrated that, if we attempted to start coding the Agents even as a proof-of-concept, lack of understanding would quickly be discovered. The FD forced the elicitation of the process, and sub-process details, required for the ReSOLV method.

- UML sequence diagrams would be a useful addition to the research process, as these successfully illustrate the end-to-end process, and would have been a useful artefact for FD, by providing context around the order in which processes occur and data is used.
- Both DFDs and UML did not naturally fit the blockchain ecosystems where multiple actors are involved. This was particularly so, where Agents do not engage specifically with each other. Although both methods are intended to be network architecture independent, in the blockchain distributed system, inter-agent communication is facilitated by the BPP as a necessary intermediary. The learning in this instance, is to explore how to better utilise DFDs and UML for blockchain software engineering.
- AOSE should be explored for the blockchain software engineering use-case. As cryptocurrency and blockchain agents become more intelligent, the new blockchain distributed system architectures (and in particular the ReSOLV architecture) are comparable to multiple agent architectures, which AOSE is specifically designed for. AOSE may be essential in developing the core ReSOLV distributed agent ecosystem, to achieve successful outcomes when developing the protocols and methods between distributed agents.
- As cryptocurrencies and blockchain technologies mature, there are a growing requirements for formal definitions to improve the quality of the software. Extensions to this research will benefit from using a Formal Methods research methodology to validate the software.

6.6 Potential Issues

A multiple agent distributed system has many design and architectural challenges requiring consideration for appropriate methodologies. The HLA (Figure 4.6) presents many possibilities, and consequently, many issues would be expected to be discovered during research, development and production. The most significant issues for ReSOLV are primarily orientated around non-functional requirements. These need to be considered during the RE phase, and will likely result in fundamental architectural changes similar to the changes, implemented in the second generation of cryptocurrencies such as Ethereum[10] and Dash[11]. However, in the context of this research, the following potential issues have been identified for discussion.

- User authorisation is a critical process in the ReSOLV method, and may be easy to attack and exploit. Using a cloud authentication service, such as OpenID or OAuth, or an authorisation service provider, such as Facebook or Google, may be required.
- The User Wallet Agent is responsible for validating the vendor application. However, the User Wallet Agent itself should also undergo validation, to detect if the Agent itself has experienced an unauthorised modification. This process is not incorporated into the RE process, and protecting the User Wallet Agent will need to be considered, to prevent compromise.
- Key management and security may become a challenge, with many public-private key pairs existing in ReSOLV, as each application has a new key pair for every transaction. Key loss or corruption will result in loss of licenses.
- The ReSOLV transaction protocol is likely to have different requirements to the Bitcoin transaction protocol, and may require extensive adaptation of an existing

[10] https://ethereum.org/
[11] https://www.dash.org/

cryptocurrency protocol. Examples include: (i) changes to the transaction output script, because Users are not spending ReSOLV tokens like bitcoins – it's a different use-case; (ii) the ReSOLV design may deliberately not support transfer of ReSOLV tokens once sent to the User Wallet Agent, as Vendors may not want software entitlement to be transferable; and (iii) the Bitcoin scripting language may be unsuitable for ReSOLV, and a customised scripting language or Turning complete transaction protocol may be required for ReSOLV transactions.

Chapter 7

Conclusion

7.1 The Software Piracy Problem

Since the late 1970s, software piracy has become commonplace, impacting software creators and affecting their ability to retain control of copyright of their works. Each decade has seen significant developments in computing technologies. New platforms such as Windows, MacOS, and Linux have become the dominant desktop platforms, whilst gaming platforms such as Xbox and PlayStation continue have grown to be mainstream household devices. In 2007, the introduction of the smartphone platform resulted in an entirely new industry being created; development of mobile Apps for the Apple IOS and Android platforms. Smartphones are the mostly widely used computing platform, with more than half the world's population using smartphones as of 2017. Software piracy has expanded across all these platforms, with methods constantly evolving to enable illegal use of software.

In addition to the developments in computing technologies, advances in communications technologies have facilitated software piracy, by providing faster delivery, portability, and mobility of software. Bandwidth continues to increase and enables faster downloads of software, with speeds of 100Mbps commonplace in 2017: up from

2.4Kbps in 1987. With the introduction of the Internet into the public domain in the late 1980's, a global communications mechanism was provided for all participants to share information. The advent of the World Wide Web in 1993 further enabled users to connect to any site to upload or download software, and supporting technologies, such as search engines, were created to solve the problem of indexing the web. As wireless technology matured, 3G and 4G high-speed wireless provided the communications platform needed for the smartphone industry to burgeon.

In an effort to reduce software piracy, software piracy protection and license provenance moved to online portals that provide software creators a platform to distribute and license software. These methods, however, do not prevent piracy; they lower the barrier to software acquisition and facilitate provenance. Software creators wanting to prevent software piracy are required to manage their own software piracy prevention methods, presenting a difficult hurdle for the smaller and independent developers, who are less likely to have the resources to manage the associated costly and complex processes. Unfortunately, threat actors continue to defeat these piracy protection and prevention methods.

Software piracy continues unabated across all platforms, creating a significant loss of revenue to software developers both large and small. For desktop platforms, many major software titles continue to be made available through piracy warez sites via BitTorrent, and the corporate vendors, despite their resources, are still finding proactive and reactive protection of their software challenging. Mobile App developers, especially game developers on Android, suffer heavy piracy due to the open nature of the Android platform and ease of copying Apps between devices. Closed systems, such as Apple iOS and the console game vendors, still experience piracy issues, but successfully discourage mainstream piracy.

The BSA piracy reports from 2004 through to 2014 shows that the economic loss

of desktop software piracy is estimated to be US$507 billion dollars, with worldwide desktop piracy rates growing from 35% to 43%. However, there is little peer reviewed research encompassing other software and hardware platforms. Gaming industry economic losses are estimated at US$74 billion in 2015, and there was no research found investigating software piracy in the smartphone App platforms or in cloud-based software piracy. This means that the researched estimate is likely to be an underestimate of the total economic loss.

Hence, a strong motivation to address software piracy has been demonstrated. The challenge that is identified is that many attempts at protecting software and allowing software creators to benefit from their copyrighted works have failed over many years. Online user authentication is revealed as the most common method for license authorisation, but does not prevent any subsequent software piracy efforts.

7.2 The Blockchain

The blockchain is a distributed ledger technology, first introduced with the creation of Bitcoin, the first cryptocurrency. It is a new data structure that is cryptographically secured and distributed across a network, typically the Internet. Blockchains are economic systems that achieve consensus among distributed nodes, allowing the transfer of digital goods without the need for centralised validation of transactions. The distributed nature of the blockchain, combined with the ability to verify transactions, has led to the development of second generation blockchain protocols that have the capability to perform new functions. NBAs have been found to provide additional application functionality built into the blockchain, allowing the software engineer to customise the blockchain protocol.

This research concludes that the characteristics of blockchain technologies suggest that NBAs can be harnessed to address existing problems, and that there is a potential

use-case for addressing the problem of software piracy prevention and provenance.

7.3 Software License Validation

The review of literature on software piracy reveals that focussing on user identity is the key to software piracy prevention. The review concludes that binding user identity to software entitlements is essential to authorising access to software and for providing an important provenance function. The drawback of user authorisation is that the user credentials can be shared between users, and that there is insufficient disincentive to sharing credentials that provide access to software. In addition, there is no protection for software-at-rest, which is vulnerable to threat actors who will attack the code structure directly to defeat prevention or protection mechanisms. These challenges help define the success criteria for an SLV mechanism.

To achieve the outcome of a cost-effective method for prevention of software piracy, it was postulated that the SLV method needs to address specific problems areas. Through this research, the requirement to address these problems areas have been determined, as described below.

It needs to be global: this requirement is supported by the determination that the end user can be located anywhere in the world, and is highly mobile. Therefore, any SLV must be global to allow ubiquitous access.

It needs to support multiple platforms: this requirement is supported by understanding that the software creator should have a single SLV mechanism for any software they create. As most ISVs release software across multiple platforms, and require a simple SLV mechanism, the need is clearly established.

It needs to be cost-effective: this requirement is supported by the understanding that smaller ISV does not have the resources or capacity to operate a SLV, and absorb

its associated operational costs. Hence SLV is mostly found on global multi-nation vendors, or through portals such as Steam, which have a narrow focus. Hence, the ISV needs a SLV with minimal overheads.

It places license authorisation in the hands of the software creator: this requirement is supported through understanding of the Software Piracy Model and the Software Piracy Vulnerability Lifecycle, where software licenses can be disclosed and copied at various stages. Enabling the software creator to allocate licenses using ReSOLV ensures end user entitlements are paid for and valid.

It protects software-at-rest: this requirement is supported through the Software Piracy Lifecycle Taxonomy, which shows the many methods of removing software protection mechanisms from software-at-rest. Any modification to software-at-rest needs to prevent the software from being executed in an unauthorised manner.

It uniquely binds user identity to software entitlements: this requirement is shown through the Software Piracy Lifecycle where end users can effect software piracy. The end user needs to be linked to the software they are authorised to execute, and should not be able to distribute software they have validly purchased.

It disincentivises credential sharing: this requirement is shown through the Software Piracy Vulnerability Lifecycle for Software-as-a-Service, where credential sharing is common, resulting in unauthorised use of software. Users need to be incentivised to maintain credential confidentiality.

It provides end user anonymity and data confidentiality: this requirement is established through the fact that the blockchain is a public ledger. Transactions need to be anonymous to maintain end user privacy, and data needs to be encrypted to maintain information confidentiality.

The broad objective for this research was to design a system that demonstrates that blockchain-based technologies can be utilised to create an SLV method. This research

has built on the Master Bitcoin Model to develop the ReSOLV SLV method. The ReSOLV SLV method leverages a user identity, combined with uniquely authorised application entitlements and software validation, to achieve a viable software piracy prevention and provenance method. For the purpose of defining the research objectives, the Hypotheses are stated as follows.

H1: To provide a Software License Validation method, a Native Blockchain Application can be applied.

H2: In respect to Software License Validation, the Native Blockchain Application enables a high level of data confidentiality and user privacy.

Research Questions were further developed to allow research to determine the validity of the hypotheses.

7.4 Limitations

There are several limitations with the ReSOLV method that have been identified. In the first instance, the scope of the cryptocurrency and blockchain ecosystem is significant, with many functional and non-functional requirements to be considered when implementing a NBA. Whilst the design of a system for SLV, such as ReSOLV, is essential, the research effort in the design phase requires significantly more time than this research permits. Further to this, a working proof of concept would demonstrate the viability of the proposed ReSOLV method, rather than establishing the feasibility through the Requirements Engineering and specification processes.

The ReSOLV method requires further research into the blockchain-based preventive mechanisms. In particular, the bootstrap method that is intended to protect the software-at-rest. This is in recognition of the fact that software-at-rest protection mechanisms will ultimately be overcome, despite separation of the bootstrap and application. With

sufficient time and motivation, an adversary will crack even complex protective mechanisms and piece together an executable, like a jigsaw, without the ReSOLV wrapper. The core issue is that all executables run in memory, and necessary information can be captured to complete the executable. Ultimately, software requires some aspect of execution or validation on a completely separate system in the ReSOLV ecosystem. The Ethereum blockchain lends itself well to this concept, with its Turing-complete smart contracts and Distributed App capability.

Investigation into protecting the ReSOLV User Agent itself is required to determine the risks to the User Agent or its database. If either of these are successfully attacked, the private keys that relate to the licenses may be stolen, resulting in the user identity and the licenses associated with the private keys being compromised.

A further limitation is that, although the user is disincentivised from sharing credentials because all their software license entitlements are disclosed, it is still possible for such sharing to occur. Additional research would be required to better understand how ReSOLV can be utilised to discourage credential sharing or encourage confidentiality. It could be possible that an adversary purchases genuine licenses, shares the private keys, and then piracy actors would be able to execute the protected software because the ReSOLV metadata can be decrypted and used.

Finally, this research focusses only on the functional requirements for the ReSOLV method. For an operational ReSOLV system, all non-functional requirements will need to be considered and addressed.

7.5 Future Work

There are many opportunities for future work in developing an NBA for the ReSOLV method. Some of these focus on the software piracy prevention and provenance, whilst others are reflect on the scale that the ReSOLV ecosystem represents.

- The most valuable future contribution is to establish a functional prototype for the ReSOLV model. This artefact will prove the fundamental SLV mechanism described in this research.
- Modelling scalability requirements for the ReSOLV method will be required to establish its viability with increasing number of users. This reflects the growth of the blockchain database, which must contain information on all license transactions, and is likely to result in blockchain bloat.
- Complementing the scalability model, a blockchain block archiving mechanism will need to be investigated, to provide controls around the size of the blockchain database. The High Level Architecture model should be further developed and revised to provide reliability and robustness across the ReSOLV ecosystem.
- As Distributed Ledger Technologies evolve, investigating the application of a non-blockchain method would be useful, to determine if blockchain non-functional requirements limitations can be overcome.
- As ReSOLV expands, in the corporate world, there will be challenges managing the user identity that is entitled to use the corporate software. This will require the capability to revoke authorisation and entitlements, even if the user is running their own personal User Wallet Agent. This may lead to the creation of a new actor such as a Corporate Provenance Agent for integration into organisations.
- The User Wallet Agent ultimately does not have to be software based. This could be an opportunity to explore using an external system or secure cloud storage.
- Most blockchains support “multi-sig”, a multiple key mechanism for authorising transactions. This would be a valuable contribution to the ReSOLV ecosystem, to protect users and prevent private key theft or unauthorised license transactions.
- The ReSOLV ecosystem presents an identity management challenge, because

user mobility will require a global mechanism to validate the user. Another future contribution would be to investigate utilising an external identity management system to provide the necessary validation.

- Investigation into a suitable Proof-of-Stake mining mechanism will be essential to determine the cryptographic security of the ReSOLV ecosystem.
- Further investigation into types of security attacks on the ReSOLV ecosystem will help identify vulnerabilities for remediation and improve confidence in the system.
- Further peer-reviewed research into the scope of piracy, and economic losses resulting from piracy across OS, mobile, gaming and cloud platforms, taking into account the gaming demographic, would be valuable in improving the understanding of the size of the piracy issues.

7.6 Conclusion

Blockchain-based technologies can be used to create innovative methods that can be applied to many use-cases that enable authorised access to data and information held in a public ledger database. Blockchain technologies can enable new processes and systems through disintermediation, and hence are considered a disruptive technology to many industries.

This research has investigated the use of a Native Blockchain Application to address the problem of software piracy prevention and provenance. It first examined the Master Bitcoin Model license provenance proof-of-concept, but identified that it was too simplistic for license provenance, in that it fails to take account of the complex nature of software licensing and distribution. Furthermore, Master Bitcoin Model provides no protection against most of the forms of software piracy as established in the literature review. The ReSOLV model was proposed to overcome the limitations of the Master

Bitcoin Model. It utilises the immutable blockchain and well established public-key cryptography methods to ensure confidentiality of proposed license data stored on the blockchain, including license key, software hash, bootstrap loader, and a digital signature field. The ReSOLV model demonstrates it provides a global method for software license provenance and software piracy protection. It thereby demonstrates that it meets the fundamental requirements to solve a global piracy problem.

Building on the ReSOLV model as the suitable method for solving the global piracy problem, this research further sought to determine which cryptocurrency provides the most suitable base for the SLV use-case, and how privacy and confidentiality of data on the blockchain could be maintained. It investigated the non-functional requirements related to the ReSOLV method, with the intention of starting an enquiry into the characteristics of existing cryptocurrency ecosystems, to determine their suitability for the ReSOLV method. Unfortunately, after initial research, the scope of non-functional requirements was subsequently deemed too large, for the purposes of this research in respect to time constraints, and consequently this research did not investigate suitable base cryptocurrencies. However, the Vendor Provence System demonstrated that data confidentiality and user privacy requirements are independent of the base cryptocurrency used. Hence, the ReSOLV model can be determined to be cryptocurrency neutral in respect to data confidentiality and privacy requirements.

The outcomes have resulted in new artefacts being created to support the hypotheses established in Section 2.8.2 on page 92, and the Research Questions presented have been satisfied as demonstrated below.

RQ1.1 is satisfied by describing how an NBA can be utilised for validating user entitlements through a ReSOLV method. The blockchain-based method provides the core mechanisms for storage and distribution of the software licenses and data used to validate the application. Licenses are protected from any copying

process, and applications will not execute if any unauthorised modification of the application is undertaken. The concept of the application bootstrap being stored in the blockchain helps protect software-at-rest, as the application cannot execute. The basis of the ReSOLV method is indeed that the application can be easily copied, as authorisation to execute is user-centric.

RQ1.2 is satisfied by showing that the ReSOLV method requirements specification are platform neutral, and that ReSOLV agents can be written for multiple platforms, such as Microsoft Windows and Android. The ReSOLV blockchain data consists of licenses, hashes and digital signatures, which are independent of the application platform. The bootstrap by definition must be operating system specific, but it is related to the application identity, and is extracted from the application code to be inserted on the blockchain. This allows a software creator to use ReSOLV to protect their software across multiple platforms.

RQ1.3 is satisfied by demonstrating that the software upgrade mechanism ensures that every hash of every authorised application and its subsequent patches are recorded on the blockchain. The ReSOLV User Wallet Agent acts as an independent authority, checking the hash of the application, based on the digitally signed hash on the blockchain, thereby identifying if the application has been modified.

RQ2.1 is satisfied through the use of asymmetric private-public key encryption, commonly used to encrypt data traversing public mediums. As the blockchain is a public ledger, any data that is placed on the blockchain is in full public view, and expected to be encrypted. Utilisation of existing ReSOLV private/public key pairs for encryption of data on the blockchain provides effective encryption of data on the blockchain, and therefore, maintains effective confidentiality.

RQ2.2 is satisfied by demonstrating that cryptocurrency methods, using private-public keys for address creation, provide anonymity and actor privacy. Anonymity and privacy in the ReSOLV method is achieved through the cryptocurrency

address creation mechanisms, each license having a source unique address from the Vendor, and each User Wallet Agent creates a new license address when requesting a license from the Vendor Provenance Agent, ensuring that the Clients cannot be identified by parties reading the blockchain.

This research concludes that the hypothesis of using the NBA for SLV is confirmed. The research further confirms the hypothesis that a high level of data confidentiality and user privacy can be attained, not only through NBA, but using other cryptocurrencies. This research therefore confirms both hypotheses as stated.

References

Acronis. (2016). *Acronis Launches Blockchain Technology Initiative with Availability of Prototype Data Authenticity Solution for Service Providers and Businesses.* Retrieved from http://www.acronis.com/en-us/business/blockchain-notary/

Adzic, G. (2011). *Specification by Example*. Manning Publications Co.

Adzic, G. (2015). *How to get the most out of Given-When-Then.* Retrieved from https://gojko.net/2015/02/25/how-to-get-the-most-out-of-given-when-then/

Agha, G. (2013). Software Engineering and Formal Methods. In *14th international conference, sefm 2016, held as part of staf 2016, vienna, austria, july 4-8, 2016, proceedings* (Vol. 8137, pp. 16–30). doi: 10.1007/978-3-319-41591-8_1

AI Blockchain. (2017). *Artificial Intelligence BlockChain Technology.* Retrieved from http://ai-blockchain.com/

Al-fedaghi, S. (2016). Design Functional Decomposition Based on Flow. In *2016 ieee international conference on systems, man, and cybernetics* (pp. 2693–2698). Budapest, Hungary.

Ali, M., Nelson, J., Labs, B., Shea, R., Labs, B., Freedman, M. J., ... Freedman, M. J. (2016). Blockstack : A Global Naming and Storage System Secured by Blockchains. *USENIX Annual Technical Conference*, 181–194. Retrieved from https://www.usenix.org/conference/atc16/technical-sessions/presentation/ali

Alqassem, I. & Svetinovic, D. (2015). Towards reference architecture for cryptocurrencies: Bitcoin architectural analysis. *Proceedings - 2014 IEEE International Conference on Internet of Things, iThings 2014, 2014 IEEE International Conference on Green Computing and Communications, GreenCom 2014 and 2014 IEEE International Conference on Cyber-Physical-Social Computing, CPS 20*(iThings), 436–443. doi: 10.1109/iThings.2014.78

Andrés, A. R. (2006). The relationship between copyright software protection and piracy: Evidence from europe. *European Journal of Law and Economics*, *21*(1), 29–51. doi: 10.1007/s10657-006-5670-5

Android marketplace hit by malware [Journal]. (2011). *Computer Fraud & Security*, *2011*(3), 3. doi: 10.1016/S1361-3723(11)70025-2

Arxan Technologies. (2015). *4th Annual State of Application Security Report: A Look Inside the Universe of Pirated Software and Digital Assets* [report].

Asongu, S. (2014). Software piracy, inequality and the poor: evidence from Africa. *Journal of Economic Studies*, *41*(4), 526–553. doi: 10.1108/JES-10-2012-0141

Assia, Y., Buterin, V., Hakim, L. & Rosenfeld, M. (2014). *Colored Coins BitcoinX.* Retrieved from https://docs.google.com/document/d/1AnkP{_}cVZTCMLIzw4DvsW6M8Q2JC0lIzrTLuoWu2z1BE/edit

Athey, S. & Stern, S. (2013). *The Nature and Incidence of Software Piracy : Evidence from Windows.* NATIONAL BUREAU OF ECONOMIC RESEARCH. Retrieved from http://www.nber.org/papers/w19755

Atom0s. (2016). *Steamless.* Retrieved 2016-01-25, from https://github.com/atom0s/Steamless

Baird, L. (2016). *Hashgraph consensus: fair, fast, byzantine fault tolerance.* Retrieved from http://www.the-blockchain.com/docs/Swirlds-Consensus-Algorithm.pdf

Baran, P. (1964). On distributed communications networks. *Communications Systems, IEEE Transactions*, *12*(1), 1–9. Retrieved from https://www.rand.ngo/content/dam/rand/pubs/papers/2005/P2626.pdf

Bashir, M., Strickland, B. & Bohr, J. (2016). What Motivates People to Use Bitcoin? BT - Social Informatics: 8th International Conference, SocInfo 2016, Bellevue, WA, USA, November 11-14, 2016, Proceedings, Part II. In E. Spiro & Y.-Y. Ahn (Eds.), (pp. 347–367). Cham: Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-47874-6{_}25 doi: 10.1007/978-3-319-47874-6_25

Bass, E., Bault, T., Baum, A., Channell, J. & Englander, S. (2014). *Disruptive Innovations II.* Citi GPS. Retrieved from https://www.citivelocity.com/citigps/ReportSeries.action?recordId=25

Baumann, A., Peinado, M. & Hunt, G. (2014). *Shielding applications from an untrusted cloud with Haven.* Broomfield, CO: Microsoft Research. Retrieved from https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-baumann.pdf

Bentov, I., Gabizon, A. & Mizrahi, A. (2014). *Cryptocurrencies without Proof of Work* [White paper].

Besen, S. M. & Raskind, L. J. (1991). An Introduction to the Law and Economics of Intellectual Property. *The journal of economic perspectives*, *5*(1), 3–27.

Bilger, J. (2006). *The Role of Cryptography in Combating Software Piracy.* Retrieved from https://courses.cs.washington.edu/courses/csep590/06wi/finalprojects/bilger.pdf

Bill and Melinda Gates Foundation. (2014). *Fighting poverty, profitably* (Tech. Rep.). Retrieved from https://docs.gatesfoundation.org/Documents/FightingPovertyProfitablyFullReport.pdf

Böhme, R., Christin, N., Edelman, B. & Moore, T. (2015). Bitcoin: Economics, Technology, and Governance. *Journal of Economic Perspectives*, *29*(2), 213–238. Retrieved from http://pubs.aeaweb.org/doi/10.1257/jep.29.2.213 doi: 10.1257/jep.29.2.213

Boneh, D., Sahai, A. & Waters, B. (2012). Functional Encryption : A New Vision for

Public-Key Cryptography. *Communications of the ACM*, *55*(11), 56–64. doi: 10.1021/ac60289a702

Bott, J. & Milkau, U. (2016). Towards a Framework for the Evaluation and Design of Distributed Ledger Technologies in Banking and Payments. *Journal of Payments Strategy & Systems*, *10*(2), 153–171.

Bradbury, D. (2013). The problem with Bitcoin. *Computer Fraud & Security*, *2013*(11), 5. doi: 10.1016/S1361-3723(13)70101-5

Bradbury, D. (2014). *Bitcoin Core Development Update #5 brings better transaction fees and embedded data.* Coindesk. Retrieved from http://www.coindesk.com/bitcoin-core-dev-update-5-transaction-fees-embedded-data/

Brhel, M., Meth, H., Maedche, A. & Werder, K. (2015). Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology*, *61*, 163–181. Retrieved from http://dx.doi.org/10.1016/j.infsof.2015.01.004 doi: 10.1016/j.infsof.2015.01.004

Brikman, Y. (2014). *Bitcoin by analogy.* Retrieved 2017-04-17, from http://www.ybrikman.com/writing/2014/04/24/bitcoin-by-analogy/

Bryans, D. (2014). Bitcoin and money laundering: mining for an effective solution. *Indiana Law Journal*, *89*(1), 441.

BSI Standards. (2011). *BSI Standards Publication Systems and software engineering — System life cycle processes.*

Business Software Alliance. (2014a). *The Compliance Gap* (Tech. Rep.). Retrieved from http://goo.gl/9WZYz6

Business Software Alliance. (2014b). *Types of Piracy* [Internet Web Page]. Retrieved from http://ww2.bsa.org/country/Anti-Piracy/What-is-Software-Piracy/TypesofPiracy.aspx?sc{_}lang=en-AU

Business Software Alliance. (2017). *Compliance and Enforcement.* Retrieved 2017-07-31, from http://www.bsa.org/anti-piracy

Business Software Alliance (BSA). (2004). *First Annual BSA And IDC Global Software Piracy Study 2004* (Tech. Rep.). Retrieved from http://www.bsa.org/{~}/media/Files/ResearchPapers/GlobalStudy/2004/IDC{_}GlobalPiracyStudy{_}2004.pdf

Business Software Alliance (BSA). (2006). *Third Annual BSA and IDC Global Software Piracy Study 2006* (Tech. Rep.). Retrieved from http://www.bsa.org/{~}/media/Files/ResearchPapers/GlobalStudy/2005/IDC{_}Global{_}Software{_}Piracy{_}Study{_}2005.pdf

Buterin, V. (2013). Introducing Ripple. *Bitcoin Magazine*. Retrieved from http://bitcoinmagazine.com/3506/introducing-ripple/

Buterin, V. (2014). Mining Pool Centralization At Crisis Levels. *Bitcoin Magazine*. Retrieved from http://bitcoinmagazine.com/9402/mining-pool-centralization-crisis-levels/

Buterin, V. (2017). *A Next-Generation Smart Contract and Decentralized Application Platform.* Retrieved from https://github.com/ethereum/wiki/wiki/White-Paper

Chabinsky, S. (2015). *The Future of Cyber Crime* (Vol. 52) (No. 3). Troy: BNP Media.

Chaitanya, K. K. (2013). Architecting the Network for the Cloud using Security Guidelines. *International Journal of Computer Applications*, *81*(8). doi: 10.5120/14035-2186

Chakravorty, A. & Rong, C. (2017). Ushare: User Controlled Social Media Based on Blockchain. *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication*, 99:1—-99:6. Retrieved from http://doi.acm.org/10.1145/3022227.3022325 doi: 10.1145/3022227.3022325

Chaudhry, P., Chaudhry, S., Stumpf, S. & Sudler, H. (2011). Piracy in cyber space: consumer complicity, pirates and enterprise enforcement. *Enterprise Information Systems*, *5*(2), 255–271. doi: 10.1080/17517575.2010.524942

Chen, X., Chu, X. & Li, Z. (2014). Improving sustainability of BitTorrent darknets. *Peer-to-Peer Networking and Applications*, *7*(4), 539–554. doi: 10.1007/s12083-012-0149-3

Chiang, O. (2011). *The Master of Online Mayhem* [Internet Web Page]. Forbes. Retrieved from http://www.forbes.com/forbes/2011/0228/technology-gabe-newell-videogames-valve-online-mayhem.html

Choi, H., Au, Y. & Liu, C. (2014). Is Digital Piracy An Enemy of the Mobile App Industry? An Empirical Study on Piracy of Mobile Apps. *AMCIS 2014 Proceedings*, 1–9. Retrieved from http://aisel.aisnet.org/amcis2014/Posters/eBusiness/7

Chuen, D. L. E. E. K. (2015). *Handbook of Digital Currency*. Saint Louis, UNITED STATES: Elsevier Science. Retrieved from http://ebookcentral.proquest.com/lib/aut/detail.action?docID=2042366

Claburn, T. (2009). Apple Expands In App Purchasing To Fight Piracy. *InformationWeek*(Generic). Retrieved from http://www.informationweek.com/apple-expands-in-app-purchasing-to-fight-piracy/d/d-id/1084052?cid=nl{_}iw{_}daily{_}html

Computer History Museum. (2016). *Computer History Museum* [Internet Web Page]. Retrieved from http://www.computerhistory.org/

Conner, K. R. (1991). Software piracy: an analysis of protection strategies. *Management Science*, *37*(2), 125–139. doi: 10.1287/mnsc.37.2.125

Courtois, N. T. (2014). *Crypto Currencies And Bitcoin* [White paper]. Retrieved from http://www.nicolascourtois.com/bitcoin/paycoin{_}may{_}2014.pdf

Courtois, N. T. & Bahack, L. (2014). *On Subversive Miner Strategies and Block Withholding Attack in Bitcoin Digital Currency*. University College London. Retrieved from http://arxiv.org/pdf/1402.1718.pdf

Cronin, G. (2002). *A Taxonomy of Methods for Software Piracy Prevention*. Auckland. Retrieved from http://www.veryquick.org/writing/piracytaxonomy.pdf

Crowder, J. A. & Friess, S. (2013). *Systems engineering agile design methodologies* (Nos. Book, Whole). New York, NY: Springer. doi: 10.1007/978-1-4614-6663-5

Danezis, G. & Meiklejohn, S. (2015). Centrally Banked Cryptocurrencies.

Davies, C. (2013). *95% Android game piracy experience highlights app theft challenge* (Vol. 2015) (No. 22/03/2015). Slashgear. Retrieved from http://www.slashgear.com/95-android-game-piracy-experience-highlights-app-theft-challenge-15282064/

DeMarines, V. (2008). Obfuscation - how to do it and how to crack it. *Network Security*, *2008*(7), 4–7. doi: 10.1016/S1353-4858(08)70085-0

Depoorter, B. (2014). What happened to video game piracy? *Communications of the ACM*, *57*(5), 33–34. Retrieved from http://dl.acm.org/citation.cfm?doid=2594413.2594289 doi: 10.1145/2594289

Dimitrijevic, S., Jovanovic, J. & Devedzic, V. (2015). A comparative study of software tools for user story management. *Information and Software Technology*, *57*(1), 352–368. doi: 10.1016/j.infsof.2014.05.012

Drachen, A., Bauer, K. & Veitch, R. W. D. (2011). Distribution of Digital Games via BitTorrent (Pre-print). *Proceedings of the 15th International Academic MindTrek Conference on Envisioning Future Media Environments - MindTrek '11*, 233. Retrieved from http://dl.acm.org/citation.cfm?doid=2181037.2181077 doi: 10.1145/2181037.2181077

Duivestein, S. & Savalle, P. (2014). *Bitcoin: It's the platform, not the currency, stupid!* Retrieved from http://thenextweb.com/insider/2014/02/15/bitcoin-platform-currency/1/

Dunn, J. (2013). *Bitcoin startup BIPS loses $1 million after DDoS heist.* Techworld. Retrieved from http://news.techworld.com/security/3490907/bitcoin-startup-bips-loses-1-million-after-ddos-heist/

Eddy, M. (2016). Crypto-Wars: Why the Fight to Encrypt Rages On. *PC Magazine*, p114–135, 22p.

Electronic Arts. (2016). *Electronic Arts EULA.* Retrieved 2016-01-22, from http://goo.gl/8Ob5Ay

Electronic Frontier Foundation. (2015). *Coders' Rights Project Reverse Engineering FAQ* (Vol. 2015) (No. 27/09/2015). Retrieved from https://www.eff.org/issues/coders/reverse-engineering-faq

European Central Bank. (2015). *Virtual Currency Schemes (2015)* (No. February). doi: ISBN:978-92-899-0862-7(online)

EUROPOL. (2014). *The Internet Organised Crime Threat Assessment* (Tech. Rep.). Europol.

Federal Bureau of Investigation. (2015). *Intellectual Property Theft* (Vol. 2015). Retrieved from https://www.fbi.gov/about-us/investigate/white{_}collar/ipr/ipr

Financial Crimes Enforcement Network. (2014). *FIN-2014-R012: Request for Administrative Ruling on the Application of FinCEN ' s Regulations to a Virtual Currency Payment System.* FinCEN.

Folsom, T. C. (1985). Is it cheaper in Hong Kong? *PC Magazine*, 215–221. Retrieved from https://goo.gl/yOrD9i

Ford, P. (2014). *Marginally useful: Bitcoin itself may not flourish as a currency, but the underlying technology is beginning to suggest valuable new applications* (Vol. 117) (No. Generic). MIT Technology Review, Inc.

FormSERA. (2012). Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA) [Welcome]. In *2012 first international workshop on formal methods in software engineering: Rigorous and agile approaches (formsera)* (pp. iii–v). doi: 10.1109/FormSERA.2012.6229781

Fortin, C. (2011). *Master Bitcoin - The Proof of Ownership* [White paper]. Retrieved from http://frozenlock.files.wordpress.com/2011/11/master-bitcoin.pdf

Fry, J. & Cheah, E. T. (2016). Negative bubbles and shocks in cryptocurrency markets. *International Review of Financial Analysis*, *47*, 343–352. Retrieved from http://dx.doi.org/10.1016/j.irfa.2016.02.008 doi: 10.1016/j.irfa.2016.02.008

F-secure. (2014). *Mobile Threat Report Q1 2014.* Retrieved from https://www.f-secure.com/documents/996508/1030743/Mobile{_}Threat{_}Report{_}Q1{_}2014.pdf

Gantz, J. F., Vavra, T., Lim, V., Soper, P., Smith, L. & Minton, S. (2015). *Unlicensed Software and Cybersecurity Threats* (White paper). BSA | The Software Alliance: BSA | The Software Alliance. Retrieved from http://www.bsa.org/{~}/media/Files/ResearchPapers/IDCMalware/FinalIDCMalwareWPJan2015.pdf

Glaser, F. & Bezzenberger, L. (2015). Beyond Cryptocurrencies - A Taxonomy Of Decentralized Consensus. In *23rd european conference on information systems (ecis 2015)* (pp. 1–18).

Gnesi, S. & Plat, N. (2015). 2nd FME Workshop on Formal Methods in Software Engineering (FormaliSE 2014). *Icse*(FormaliSE), 977–978. doi: 10.1109/ICSE.2015.313

Goldman, E. (2004). Warez trading and criminal copyright infringement. *Journal of the Copyright Society of the U.S.A*, *51*(2), 395.

Goode, S. & Cruise, S. (2006). What Motivates Software Crackers? *Journal of business ethics*, *65*(2), 173–201. doi: 10.1007/s10551-005-4709-9

Goodin, D. (2014). *Apple's "in-app purchase" service for iOS bypassed by Russian hacker | Ars Technica* [Internet Web page]. Retrieved from http://arstechnica.com/security/2012/07/ios-in-app-purchase-service-hacked/

Gregor, S. & Hevner, A. R. (2013). Positioning and Presenting Design Science Research for Maximum Impact. *MIS Quarterly*, *37*(2), 337–355. doi: 10.2753/MIS0742-1222240302

Gurulian, I., Markantonakis, K., Cavalaro, L. & Mayes, K. (2016). You can't touch this: Consumer-centric android application repackaging detection. *Future Generation Computer Systems*, *65*, 1–9. Retrieved from http://dx.doi.org/

10.1016/j.future.2016.05.021 doi: 10.1016/j.future.2016.05.021

Haley, A. (2014). *How Bad is App Piracy Really?* Retrieved 2015-12-12, from http://www.multipie.co.uk/2014/04/bad-app-piracy-really/

Halford, R. (2014). *Gridcoin: Crypto-Currency using {B}erkeley {O}pen {I}nfrastructure {N}etwork {C}omputing {G}rid as a {P}roof {O}f {W}ork* [White paper]. Retrieved from http://www.gridcoin.us/images/gridcoin-white-paper.pdf

Han, K. & Shon, T. (2014). Software authority transition through multiple distributors. *The Scientific World Journal*, *2014*, 295789. doi: 10.1155/2014/295789

Hanley, B. P. (2013). *The False Premises and Promises of Bitcoin.* Retrieved from http://arxiv.org/pdf/1312.2048.pdf

Herbert, J. & Litchfield, A. (2015). A Novel Method for Decentralised Peer-to-Peer Software License Validation Using Cryptocurrency Blockchain Technology. In *38th australasian computer science conference (acsc 2015)* (pp. 27–30). Sydney.

Herbert, J. & Stabauer, M. (2015). Bitcoin \& Co: An Ontology for Categorising Cryptocurrencies. In *M-sphere: Book of papers* (pp. 45–55). Dubrovnik.

Hétu, D. D., Morselli, C. & Leman-Langlois, S. (2012). Welcome to the Scene: A Study of Social Organization and Recognition among Warez Hackers. *Journal of Research in Crime and Delinquency*, *49*(3), 359–382. doi: 10.1177/0022427811420876

Hevner, A. R., March, S. T., Park, J. & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75–105. Retrieved from http://dblp.uni-trier.de/rec/bibtex/journals/misq/HevnerMPR04 doi: 10.2307/25148625

Hinchey, M., Jackson, M., Cousot, P., Cook, B., Bowen, J. P. & Margaria, T. (2008, sep). Software Engineering and Formal Methods. *Communications of the ACM*, *51*(9), 54–59.

Hochstein, M. (2014). Why bitcoin matters for bankers. *American Banker*, *March 2014*. Retrieved from http://goo.gl/lYRycw

Holsapple, C. W. ., Iyengar, D., Jin, H. & Rao, S. (2008). Parameters for Software Piracy Research. *The Information Society*, *24*(4), 199–218. doi: 10.1080/01972240802189468

Hull, E., Jackson, K. & Dick, J. (2011). *Requirements Engineering*. doi: 10.1007/978-1-84996-405-0

IEEE. (1998). *IEEE Recommended Practice for Software Requirements Specifications*.

Iivari, J. & Venable, J. (2009). Action Research and Design Science Research. *Action Research*, 1–13.

Im, J. H. & Van Epps, P. D. (1992). Software piracy and software security measures in business schools. *Information & Management*, *23*(4), 193–203. doi: 10.1016/0378-7206(92)90044-G

Ionescu, D. (2010). *Microsoft Bans Up to One Million Users From Xbox Live.* PC World. Retrieved 2016-01-26, from http://www.pcworld.com/article/182010/xbox{_}users{_}banned.html

Irvine, D. (2014). *MaidSafe.net announces project SAFE to the community.* MaidSafe. Retrieved from https://github.com/maidsafe/Whitepapers/blob/master/Project-Safe.md

ISO/IEC. (1996). *ISO 9660: Color books of standards.* Retrieved from http://www.cdfs.com/cdfs-color-books.html

Jaramillo, D., Nguyen, D. V. & Smart, R. (2016). Leveraging microservices architecture by using Docker technology. *Conference Proceedings - IEEE SOUTHEASTCON*, *2016-July*, 0–4. doi: 10.1109/SECON.2016.7506647

Kammerstetter, M., Platzer, C. & Wondracek, G. (2012). Vanity, cracks and malware: insights into the anti-copy protection ecosystem. In (pp. 809–820). ACM. doi: 10.1145/2382196.2382282

Kang, H., Le, M. & Tao, S. (2016). Container and microservice driven design for cloud infrastructure DevOps. *Proceedings - 2016 IEEE International Conference on Cloud Engineering, IC2E 2016: Co-located with the 1st IEEE International Conference on Internet-of-Things Design and Implementation, IoTDI 2016*, 202–211. doi: 10.1109/IC2E.2016.26

Karas, S. (2001). Sony Computer Entertainment Inc. v Connectix Corp. *Berkeley Technology Law Journal*, *16*(1), 1–8. Retrieved from http://scholarship.law.berkeley.edu/cgi/viewcontent.cgi?article=1299{&}context=btlj doi: 10.15779/Z38S10B

Kautz, K. (2011). Investigating the Design Process: Participatory Design in Agile Software Development. *Information Technology & People*, *24*(3), 217–235. Retrieved from http://www.emeraldinsight.com/10.1108/09593841111158356 doi: 10.1108/09593841111158356

Keith, J. B., Nick, D., Peter, K., Veena, P., Likhit, W. & Wallis. (2016). Leading the pack in blockchain banking. *IBM Research*, 1–20. Retrieved from https://www.hyperledger.org/wp-content/uploads/2016/10/Leading-the-pack-in-blockchain-banking-1.pdf

KeyesLabs. (2010). *A Global Piracy Heatmap.* Retrieved from http://keyeslabs.com/joomla/projects/auto-app-licensing/152-a-global-piracy-heat-map

KeyesLabs. (2013). *Android – The Perfect Piracy Storm.* Retrieved from http://keyeslabs.com/joomla/blogs/i-think-im-becoming-an-android/136-android-the-perfect-piracy-storm

Khan, A. u. R. A. N., Othman, M., Ali, M. & Madani, S. A. (2014). Pirax: framework for application piracy control in mobile cloud environment. *The Journal of Supercomputing*, *68*(2), 753–776. doi: 10.1007/s11227-013-1061-1

Khandelwal, S. (2015). *Hacker confirms Playstation 4 Jailbreak Exploit cloud open doors for pirated games.* The Hacker News. Retrieved 2016-01-26, from http://thehackernews.com/2015/12/sony-ps4-playstation-jailbreak.html

Kigerl, A. C. (2013). Infringing Nations: Predicting Software Piracy Rates, BitTorrent Tracker Hosting, and P2P File Sharing Client Downloads Between Countries. *International Journal of Cyber Criminology U6*, *7*(1), 62.

Kirkpatrick, K. (2017). Financing the dark web. *Communications of the ACM*, *60*(3), 21–22. Retrieved from http://dl.acm.org/citation.cfm?doid=3055102.3037386 doi: 10.1145/3037386

Kondor, D., Pósfai, M., Csabai, I. & Vattay, G. (2014). *Do the rich get richer? An empirical analysis of the bitcoin transaction network* (Vol. 9) (No. 2). United States. doi: 10.1371/journal.pone.0086197

Kretschmer, M. (2003). Digital copyright: the end of an era. *European Intellectual Property Review*, *25*(8), 333–341.

Kumar, M. (2017). *A Typo in Zerocoin's Source Code helped Hackers Steal ZCoins worth $585,000.* Retrieved from http://thehackernews.com/2017/02/zcoin-zerocoin-typo.html

Lande, R. H. & Sobin, S. M. (1996). Reverse engineering of computer software and U.S. antitrust law. *Harvard Journal of Law & Technology*, *9*(2), 237.

Laplante, P. A. (2014). *Requirements Engineering for Software and Systems* (Second edi ed.). Boca Raton : CRC/Taylor & Francis.

Lebo, A. (2014). *Implementation of a decentralized, transferable, and open software license system using the Bitcoin protocol.* Retrieved from https://github.com/fisher-lebo/dissent

Ledra Capital. (2014). *The Mega-Master Blockchain* List* (Vol. 2014). Retrieved from http://ledracapital.com/blog/2014/3/11/bitcoin-series-24-the-mega-master-blockchain-list

Lejacq, Y. (2013). *Microsoft Threatens Lifetime Xbox Live Ban To Pirates After 'Gears of War: Judgement' Leaked Online.* International Business Times. Retrieved 2016-01-26, from http://www.ibtimes.com/microsoft-threatens-lifetime-xbox-live-ban-pirates-after-gears-war-judgement-leaked-online-1093418

Levy, Y., Levy, Y. & Ellis, T. J. (2011). A Guide for Novice Researchers on Experimental and Quasi-Experimental Studies in Information Systems Research A Guide for Novice Researchers on Experimental and Quasi-Experimental Studies in Information Systems Research. *Interdisciplinary Journal of Information, Knowledge, and Management*, *6*. Retrieved from http://www.ijikm.org/Volume6/IJIKMv6p151-161Levy553.pdf

Lewis, R. (2016). *Five blockchain development challenges for legacy organisations.* Retrieved 2017-05-02, from https://medium.com/yope-chain/five-blockchain-development-challenges-for-legacy-organisations-e0f57e6b808c

Lu, J. & Weber, I. (2009). Internet Software Piracy in China: A User Analysis of Resistance to Global Software Copyright Enforcement. *Journal of International and Intercultural Communication*, *2*(4), 298–317. doi: 10.1080/17513050903177300

Lynley, M. (2011). *Did Sony shut down PSN to prevent "extreme" piracy?* VentureBeat. Retrieved 2016-01-26, from http://venturebeat.com/2011/04/25/playstation-network-outage-piracy/

Maesa, D. D. F., Marino, A. & Ricci, L. (2016). Uncovering the Bitcoin Blockchain: An Analysis of the Full Users Graph. *2016 IEEE International Conference on Data*

Science and Advanced Analytics (DSAA), 537–546. Retrieved from http://ieeexplore.ieee.org/document/7796940/ doi: 10.1109/DSAA.2016.52

Mainelli, M. (2018). Blockchain Will Help Us Prove out Identities in a Digital World. *Harvard Business review*(March), 2–7.

Marian, O. (2014). A Conceptual Framework for the Regulation of Cryptocurrencies. *The University of Chicago Law Review*, *2423461*(forthcoming), 53–68.

Martin, K. (2014). Regulating Code: Good Governance and Better Regulation in the Information Age, by Ian Brown and Christopher Marsden. Cambridge, Mass.: MIT Press, 2013. ISBN: 978-0262018821. *Business Ethics Quarterly*, *24*(04), 624–627. Retrieved from https://www.cambridge.org/core/product/identifier/S1052150X00013282/type/journal{_}article doi: 10.5840/beq201424420

Maude, T. & Maude, D. (1984). Hardware protection against software piracy. *ACM*, *27*(Generic), 950–959. doi: 10.1145/358234.358271

Mazières, D. (2015). *The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus.* Retrieved from https://www.stellar.org/papers/stellar-consensus-protocol.pdf

McCook, H. (2014). *An Order-of-Magnitude Estimate of the Relative Sustainability of the Bitcoin Network.* Retrieved from http://goo.gl/M8741r

Mehdi, K.-P. (2013). *Dictionary of Information Science and Technology (2nd Edition).* Hershey, PA, USA: IGI Global. doi: 10.4018/978-1-4666-2624-9

Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G. M. & Savage, S. (2013). A fistful of bitcoins. *A Fistful of Bitcoins: Characterizing Payments Among Men with No Names*, *59*(4), 127–140. Retrieved from http://dl.acm.org/citation.cfm?id=2504730.2504747 doi: 10.1145/2504730.2504747

Meshkova, E., Riihijärvi, J., Petrova, M. & Mähönen, P. (2008). A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. *Computer Networks*, *52*(11), 2097–2128. Retrieved from http://linkinghub.elsevier.com/retrieve/pii/S138912860800100X doi: 10.1016/j.comnet.2008.03.006

Miers, I., Garman, C., Green, M. & Rubin, A. D. (2013). Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *2013 ieee symposium on security and privacy* (pp. 397–411). IEEE. doi: 10.1109/SP.2013.34

Millar, J. (2017). *Enterprise Ethereum Alliance - Vision paper.* Retrieved from https://entethalliance.atlassian.net/wiki/download/attachments/37151/EntEthVision-v3.1-24February2017.pdf?version=1{&}modificationDate=1488946292762{&}cacheVersion=1{&}api=v2

Molina-Solana, M., Birch, D. & Guo, Y.-k. (2017). Improving data exploration in graphs with fuzzy logic and large-scale visualisation. *Applied Soft Computing*, *53*, 227–235. Retrieved from //www.sciencedirect.com/science/article/pii/S1568494616306731 doi: http://dx.doi.org/10.1016/j.asoc.2016.12

.044

Mooers, C. N. (1977). Preventing Software Piracy. *Computer*, *29*(3), 29–30. doi: 10.1109/C-M.1977.217671

Moon, K. (2009). *IBM v Computer Imports [1989] 2 NZLR 395.* AJ Park. Retrieved from http://www.ajpark.com/media/98134/the{_}nature{_}of{_}computer{_}programmes.pdf

Moores, T. & Dhillon, G. (2000). Software piracy: a view from Hong Kong. *Commun. ACM*, *43*(12), 88–93. doi: 10.1145/355112.355129

Moy, E. C. (2014). *The Currency Revolution, Courtesy Of Bitcoin.* Retrieved from http://edmoy.com/the-currency-revolution-courtesy-of-bitcoin/

Munter, J. (2016). *Is Bitcoin the Next Big Thing in the Fight Against Piracy?* Copyright Alliance. Retrieved 2017-04-08, from http://copyrightalliance.org/ca{_}post/bitcoin-next-big-thing-fight-piracy/

Nakamoto, S. (2008, jan). *Bitcoin: A peer-to-peer electronic cash system* (Vol. 1) [White paper]. Retrieved from https://bitcoin.org/bitcoin.pdf

Naumovich, G. & Memon, N. (2003). Preventing piracy, reverse engineering, and tampering. *Computer*, *36*(7), 64–71. doi: 10.1109/MC.2003.1212692

Netmarketshare. (2015). *Netmarketshare* [Internet Web Page]. Retrieved from http://netmarketshare.com/

Newman, J. (2013). Illegal deposit: Game preservation and/as software piracy. *Convergence: The International Journal of Research into New Media Technologies*, *19*(1), 45–61. Retrieved from http://con.sagepub.com/content/19/1/45.abstract doi: 10.1177/1354856512456790

Nielsen, J. (2014). *Nielsen's Law of Internet Bandwidth.* Nielsen Norman Group. Retrieved 2016-04-12, from https://www.nngroup.com/articles/law-of-bandwidth/

Nill, A. & Shultz, C. J. (2009). Global software piracy: Trends and strategic considerations. *Business Horizons (2009)*, *52*, 289–298. doi: 10.1016/j.bushor.2009.01.007

Noyen, K., Volland, D., Wörner, D. & Fleisch, E. (2014). When Money Learns to Fly: Towards Sensing as a Service Applications Using Bitcoin. , 6. Retrieved from http://arxiv.org/abs/1409.5841

Nunamaker Jr., J. & Chen, M. (1990). Systems development in information systems research. *Twenty-Third Annual Hawaii International Conference on System Sciences*, *iii*(3), 89–106. doi: 10.1109/HICSS.1990.205401

NXT Community. (2014). *Nxt Whitepaper.* Retrieved from https://nxtwiki.org/wiki/Whitepaper:Nxt

Oberhauser, A. (2015). *Decentralized Public Ledger as Enabler for the Gift Economy at Scale.* University of Amsterdam. Retrieved from https://goo.gl/ajvqZf

Ois, S. N. A., Sherif, F. F. & Gamal, I. S. (2014). MCSAuth: A New Authentication Mechanism for Cloud Systems. *International Journal of Computer Applications*, *88*(15). doi: 10.5120/15428-3934

Oxford Dictionary. (2016). Retrieved from http://www.oed.com/

Palmer, B. P. (2014). *Anonymously Establishing Digital Provenance in Reseller Chains* (thesis, Victoria University Of Wellington, Wellington). Retrieved from http://researcharchive.vuw.ac.nz/xmlui/handle/10063/2281

Panzarino, M. (2013). *iOS App Piracy Continues, Now Without Jailbreak.* Retrieved 2015-12-12, from http://thenextweb.com/apple/2013/01/01/low-down-dirty-iphone-app-pirates/

Papas, N., O'Keefe, R. M. & Seltsikas, P. (2012). The action research vs design science debate: reflections from an intervention in eGovernment. *European Journal of Information Systems*, *21*(2), 147–159. doi: 10.1057/ejis.2011.50

Parliament, E. & Council, E. U. (2001). Directive 2001/29/EC. *Official Journal of the European Union*, *L167*(February), 12–25.

Patel, S. J. & Pattewar, T. M. (2014). Software birthmark based theft detection of JavaScript programs using agglomerative clustering and Frequent Subgraph Mining. *International Conference on Embedded Systems, ICES 2014*, 63–68. doi: 10.1109/EmbeddedSys.2014.6953052

Peck, M. (2017). *TalkComputingNetworks Corporate Titans Unite to Build an Enterprise Version of the Ethereum Blockchain.* Retrieved from http://spectrum.ieee.org/tech-talk/computing/networks/enterprise-ethereum-alliance-launches

Peffers, K., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. doi: 10.2753/MIS0742-1222240302

Percival, C. (2009). *Stronger key derivation via sequential memory-hard functions.* Retrieved from http://www.tarsnap.com/scrypt/scrypt.pdf

Peteanu, R. (2014). *Fraud Detection in the World of Bitcoin.* Coin Publishing Ltd. Retrieved from http://bitcoinmagazine.com/11599/fraud-detection-world-bitcoin/

Plassaras, N. A. (2013). Regulating Digital Currencies: Bringing Bitcoin within the Reach of the IMF. *Chicago Journal of International Law*, *14*(1), 377.

Ployhar, M. (2012). *Gaming Piracy - Separating Fact from Fiction* [Internet Web Page]. Retrieved from https://software.intel.com/en-us/blogs/2012/09/22/gaming-piracy-separating-fact-from-fiction

Porter, R. (2015). *Digital currencies: response to the call for information* (No. March). Retrieved from http://www.pwc.co.uk/en{_}UK/uk/assets/pdf/hm-treasury-call-for-information-digital-currencies.pdf

Prisco, G. (2015a). The New Stellar Consensus Protocol Could Permit Faster and Cheaper Transactions. *Bitcoin Magazine*. Retrieved from https://bitcoinmagazine.com/20044/new-stellar-consensus-protocol-permit-faster-cheaper-transactions/

Prisco, G. (2015b). Peter Diamandis: Blockchain Technology Will Enable

Extraordinary Transform. *Bitcoin Magazine*. Retrieved from https://bitcoinmagazine.com/20407/peter-diamandis-blockchain-technology-will-enable-extraordinary-transformation/

Radcliffe, M. (1999). Digital Millennium Copyright Act: Forging the Copyright Framework for the Internet: First Steps. *Journal of Internet Law*, *2*(9), 1.

Rapoza, K. (2012). *In China, Why Piracy Is Here To Stay* (Vol. 2015). Forbes. Retrieved from http://www.forbes.com/sites/kenrapoza/2012/07/22/in-china-why-piracy-is-here-to-stay/

Rasch, A. & Wenzel, T. (2015). The impact of piracy on prominent and non-prominent software developers. *Telecommunications Policy*, *39*(8), 735–744. Retrieved from http://linkinghub.elsevier.com/retrieve/pii/S0308596114001839 doi: 10.1016/j.telpol.2014.11.003

Ravenscraft, E. (2012). *Just How Bad Is App Piracy On Android Anyway? Hint: We're Asking The Wrong Question.* Retrieved from http://www.androidpolice.com/2012/07/31/editorial-just-how-bad-is-app-piracy-on-android-anyways-hint-were-asking-the-wrong-question/

Rivest, R. L., Shamir, A. & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2), 120–126. Retrieved from http://portal.acm.org/citation.cfm?doid=359340.359342 doi: 10.1145/359340.359342

Rosenfeld, M. (2012). *Overview of Colored Coins.* Retrieved from https://bitcoil.co.il/BitcoinX.pdf

Schaefer, I. & Hähnle, R. (2011). Formal Methods in SPL. *IEEE Computer*, 82–85.

Schön, E.-m., Thomaschewski, J. & Escalona, M. J. (2016). Agile Requirements Engineering: A Systematic Literature Review. *Computer Standards & Interfaces*, *49*, –. doi: http://dx.doi.org/10.1016/j.csi.2016.08.011

Senior, T. (2012). *PC gaming has "around a 93-95% piracy rate" claims Ubisoft CEO* [Internet Web Page]. Retrieved from http://www.pcgamer.com/pc-gaming-has-around-a-93-95-per-cent-piracy-rate-claims-ubisoft-ceo/

Sharma, S., Sharma, C. S. & Tyagi, V. (2015). Plagiarism detection tool "Parikshak". *2015 International Conference on Communication, Information & Computing Technology (ICCICT)*, 1–7. doi: 10.1109/ICCICT.2015.7045739

Shea, R. (2017). *Blockstack.* Retrieved 2017-04-17, from https://blockstack.org/intro

Shehory, O. M. & Sturm, A. (2014). *Agent-oriented software engineering : reflections on architectures, methodologies, languages, and frameworks.* Berlin : Springer, [2014].

Sheinblatt, J. S. (2014). The WIPO Copyright Treaty. *Berkeley Tech. L.J.*, *13*(1), 535–550. Retrieved from http://scholarship.law.berkeley.edu/btlj/vol13/iss1/34

Shirriff, K. (2014). *Hidden surprises in the Bitcoin blockchain and how they are stored.* Retrieved from http://www.righto.com/2014/02/ascii

-bernanke-wikileaks-photographs.html
Sigi, G. (2010). Exploring the supply of pirate software for mobile devices: An analysis of software types and piracy groups. *Information Management & Computer Security & Computer Security*, *18*(4), 204–225. doi: 10.1108/09685221011079171
Silver, M. S., Markus, M. L. & Beath, C. M. (1995). The Information Technology Interaction Model: A Foundation for the MBA Core Course. *MIS Quarterly*, *19*(3), 361–390. doi: 10.2307/249600
Singh, S., Jeong, Y. S. & Park, J. H. (2016). A survey on cloud computing security: Issues, threats, and solutions. *Journal of Network and Computer Applications*, *75*, 200–222. Retrieved from http://dx.doi.org/10.1016/j.jnca.2016.09.002 doi: 10.1016/j.jnca.2016.09.002
Smith, D. (2015). *Android Still Has A Massive Piracy Problem* [Internet Web Page]. Business Insider Australia. Retrieved from http://www.businessinsider.com.au/android-piracy-problem-2015-1
Solon, O. (2013). *BitTorrent study challenges videogame piracy misconceptions (Wired UK).* Retrieved 2016-01-18, from http://www.wired.co.uk/news/archive/2013-05/15/bittorrent-gaming-study
Sommerville, I. (2010). *Software Engineering, Ninth Edition* (Ninth Edit ed.). doi: 10.1111/j.1365-2362.2005.01463.x
Sony. (2016a). *Information on Banned Accounts and Consoles.* Retrieved 2016-02-29, from https://support.us.playstation.com/articles/en{_}US/KC{_}Article/Information-on-Banned-Accounts-and-Consoles/?
Sony. (2016b). *Playstation Software Usage Terms.* Retrieved 2016-01-22, from https://www.playstation.com/en-nz/legal/software-usage-terms/
Sprankel, S. (2013). *Technical Basis of Digital Currencies.*
Steam. (2015). *Steam Hardware & Software Survey* [Internet Web Page]. Retrieved from http://store.steampowered.com/hwsurvey?platform=combined
Steam. (2016). *Steam Anti-Piracy Suite.* Retrieved from https://www.steampowered.com/steamworks/publishingservices.php
SteamRE. (2016). *Open-Steamworks.* Retrieved 2016-01-25, from https://github.com/SteamRE/open-steamworks
Suduc, A.-M., Bizoi, M. & Filip, F. G. (2009). *Ethical Aspects on Software Piracy and Information and Communication Technologies Misuse* (Vol. 42) (No. 25). IFAC. Retrieved from http://dx.doi.org/10.3182/20091028-3-RO-4007.00008 doi: 10.3182/20091028-3-RO-4007.00008
Suhler, P. A., Bagherzadeh, N., Malek, M. & Iscoe, N. (1986). Software Authorization Systems. *IEEE Software*, *3*(5), 34–41. doi: 10.1109/MS.1986.234396
Swanson, S. D. (2010). The Digital Millennium Copyright Act and the Iphone: An Unnecessary Proceeding. *Journal of Internet Law*, *14*(August), 3–6.
Techopedia. (2015). *Software Piracy* (Vol. 2015) (Internet Web Page No. 03/09/2015).

Retrieved from https://www.techopedia.com/definition/4361/software-piracy

Thurimella, A. K. (2014). Using Case-Based Methods in an Experimental Design : A Mixed-Method Approach for Evaluating Collaboration-Intensive Software Modeling Approaches. *Journal of Case Research*, *V*(02), 80–91.

Tsotsorin, M. (2013). *Piracy and Video Games: Is There a Light at the End of the Tunnel?* Retrieved from http://works.bepress.com/cgi/viewcontent.cgi?article=1002{&}context=maxim{_}tsotsorin

Ubisoft. (2016). *Ubisoft EULA*. Retrieved 2016-01-22, from https://legal.ubi.com/eula/en-US

Underwood, S. (2016). Blockchain beyond bitcoin. *Communications of the ACM*, *59*(11), 15–17. Retrieved from http://dl.acm.org/citation.cfm?doid=3013530.2994581 doi: 10.1145/2994581

UNESCO. (2015). *World Anti-Piracy Observatory* (Vol. 2015) (Internet Web Page No. 03/09/2015). Retrieved from http://portal.unesco.org/culture/en/ev.php-URL{_}ID=39397{&}URL{_}DO=DO{_}TOPIC{&}URL{_}SECTION=201.html

University of Sheffield. (2016). *CFP: Blockchain Applications In Artificial Intelligence And Cognitive Science.* Retrieved from http://blockchainstudies.org/BlockchainCogAI.html

Van der Sar, E. (2009). MAC vs PC: The Pirate Edition. *Torrentfreak*. Retrieved from https://torrentfreak.com/why-mac-users-are-better-pirates-090206/

Veerubhotla, R. S. & Saxena, A. (2011). A DRM framework towards preventing digital piracy. *Proceedings of the 2011 7th International Conference on Information Assurance and Security, IAS 2011*, 1–6. doi: 10.1109/ISIAS.2011.6122785

Wagner, S. C. & Sanders, G. L. (2001). Considerations in ethical decision-making and software piracy. *Journal of business ethics*, *29*(1-2), 161–167.

Wang, C.-w., Cheng, M., Cho, Y. & Wang, C.-w. (2015). Combating Software Piracy in Public Clouds. *Computer*, *48*(10), 88–91. doi: 10.1109/MC.2015.317

Williams, L. (2012). What agile teams think of agile principles. *Communications of the ACM*, *55*(4), 71. Retrieved from http://dl.acm.org/citation.cfm?doid=2133806.2133823 doi: 10.1145/2133806.2133823

Winters, T. (2014). *Proof of Burn and the Counterparty Approach.* Retrieved from http://bitscan.com/articles/proof-of-burn-and-the-counterparty-approach

Wolff, S. (2012). Scrum Goes Formal : Agile Methods for Safety-Critical Systems. In *2012 first international workshop on formal methods in software engineering: Rigorous and agile approaches (formsera)* (pp. 23–29).

Wood, G. (2016). *Ethereum: A Secure Decentralised Generalised Transaction Ledger EIP-150 Revision.* Retrieved from http://gavwood.com/Paper.pdf?TB{_}iframe=true{&}width=288{&}height=432

Wood, G., Zamfir, V. & Coleman, J. (2015). *Notes on Scalable Blockchain Protocols.*

Wooldridge, M. & Jennings, N. R. (1995). Intelligent Agents : Theory and Practice. *The Knowledge Engineering Review*, *10*(2), 115–152.

World Economic Forum. (2015). *Deep Shift Technology Tipping Points and Societal Impact* (Tech. Rep. No. September). World Economic Forum.

Xiaochao, Q. (2014). A Bitcoin system with no mining and no history transactions: Build a compact Bitcoin system.

Xiaosong, L. & Kai, H. (2009). Collusive Piracy Prevention in P2P Content Delivery Networks. *Computers, IEEE Transactions on*, *58*(7), 970–983. doi: 10.1109/TC.2009.26

Yin-Leng, T., Wee Teck, T., May, O. L. & Schubert Foo, S.-B. (2010). An Exploratory Study of Determinants and Corrective Measures for Software Piracy and Counterfeiting in the Digital Age. *Computer and Information Science*, *3*(3), 30. doi: 10.5539/cis.v3n3p30

Yulong Zhang, Zhaofeng Chen, Y. K. (2015). *Guaranteed Clicks: Mobile App Company Takes Control of Android Phones.* Retrieved from `https://www.fireeye.com/blog/threat-research/2015/09/guaranteed{_}clicksm.html`

Zave, P. (1997). Classification of Research Efforts in Requirements Engineering. *Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95)*, *29*(4), 315–321. doi: 10.1109/ISRE.1995.512563

Zimmermann, K. A. (2012). *Internet History Timeline: ARPANET to the World Wide Web* (Vol. 2015) (No. 18/04/2015). LiveScience. Retrieved from `http://www.livescience.com/20727-internet-history.html`

Appendix A

A.1 Definitions

AD	Active Directory
AOSE	Agent Oriented Systems Engineering
API	Application Programming Interface
AS	Application System
BSA	Business Software Alliance
BPP	Blockchain Peer Pool
CEG	Custom Executable Generation
COTS	Commercial off-the-shelf (software)
CUS	Client User System
DApps	Distributed Apps
DCMA	Digital Millennium Copyright Act
DDS	Digital Distribution Services
DFD	Data Flow Diagram
DNS	Domain Name Service
DRM	Digital Rights Management
DSR	Design Science Research
EBA	External Blockchain Application
ECDSA	Elliptic Curve Digital Signature Algorithm
EULA	End User License Agreement
GWT	Given-When-Then
HLA	High Level Architecture
IS	Information Systems
ISV	Independent Software Vendor
LDAP	Lightweight Directory Access Protocol
M2C	Machine to Consumer

M2M	Machine to Machine
MDM	Master Bitcoin Model
NBA	Native Blockchain Application
OS	Operating System
PApp	Published App
PC	Personal Computer
RA	Reference Architecture
RE	Requirements Engineering
ReSOLV	The blockchain-based SLV method being researched
RVC	Regulated Virtual Currency
SaaS	Software-as-a-Service
SAFE	Secure Access For Everyone
SLV	Software License Validation
SoV	States Of Vulnerability
SPVL	Software Piracy Vulnerability Lifecycle
TAP	Transaction and Application Platform
TCP/IP	Transmission Control Protocol / Internet Protocol
TOC	Transaction-only Cryptocurrency
TTP	Tagged Transaction Protocol
TRIPS	Trade Related Aspects of Intellectual Property Rights
UML	Unified Modelling Language
USB	Universal Serial Bus
VPS	Vendor Provenance System
WIPO	World Intellectual Property Organization

A.2 Bitcoin Protocol Learnings

There were several learnings in respect to the Bitcoin protocol that resulted from the detail and pseudocode required to produce the data flow diagrams. The underlying bitcoin transaction process requires a careful understanding of the elements involved to determine the data flows. Furthermore, although the data flow diagrams are independent of system architecture, data relationships, and state, their relevance became important in the context of understanding the bitcoin ecosystem. This was determined because the concept of a “chain of digital signatures” requires potentially different sets of data from earlier transactions that are stored by the Miners in the blockchain public ledger.

In summary, the bitcoin transaction process between two actors has two parts, inputs

and outputs, that must balance for each transaction as discussed in Chapter 2.5.5 on page 73. Expanding on this, when an actor wants to send a bitcoin of some value to another actor, the sending actor transmits a message using the Bitcoin protocol to the Bitcoin peer group (the Miners) to update the public ledger (the blockchain), and awaits confirmation of the transaction to be announced to the Bitcoin network. During the creation of the data flow diagrams, this simplistic explanation required further investigation to unravel the mechanics of the transaction process and determine what data is utilised during the end to end of the transaction process. The Bitcoin protocol was used as the basis for describing how ReSOLV might interact within a ReSOLV ecosystem. Despite many available sources of information describing the Bitcoin protocol, the information is often confusing, as references to which public key, private key, and address are often omitted from explanations. The best deconstruction of a Bitcoin ecosystem transaction was provided by Shirriff (2014), who manually created a bitcoin transaction using Python coding language.

A concise explanation of the Bitcoin transaction process is described below, with some precursory information provided that establishes the basic elements of a bitcoin transaction. For clarity of the explanation, Alice and Bob are used as the primary actors, where Alice is sending bitcoins to Bob. Jim, Kate and Linda are secondary actors, used to demonstrate the how Alice received her bitcoins and the authenticity of the transactions. Figure A.1 shows the various elements that are used to construct the bitcoin transaction, and these are explained in the following sections.

A.2.1 Precursor

In the Bitcoin cryptocurrency ecosystem, bitcoins do not exist as an entity. Bitcoins are simply signed data on a public ledger (the blockchain), where one actor can digitally sign data over to another actor: hence the original definition of a bitcoin as “a chain

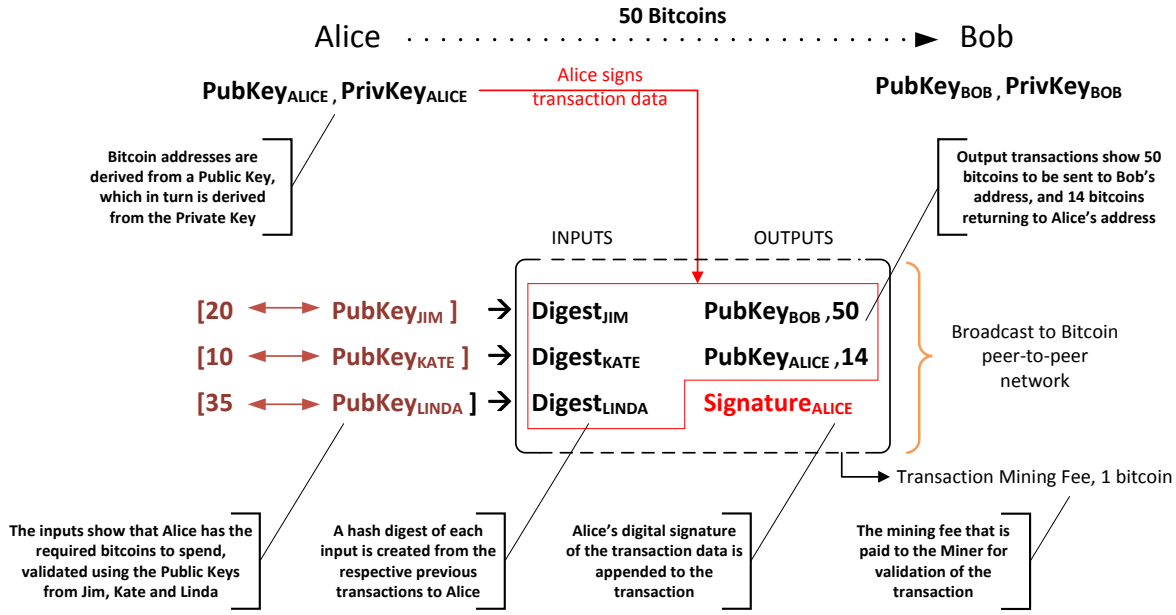


Figure A.1: Bitcoin Transaction

of digital signatures" (Nakamoto, 2008). Essentially, what one owns in the Bitcoin cryptocurrency ecosystem is transaction receipts.

Each actor has at least one bitcoin address, which is derived from the actor's public key, which in turn is created using an Elliptic Curve Digital Signature Algorithm (ECDSA) that generates a private-public key pair. The algorithm to derive the public key is itself public, meaning than anyone can use a public key to validate that a specific bitcoin address has the correct association with that specific public key. Actors have digital wallets that store the private key, and may well store the associated public key, bitcoin addresses and the data that represents bitcoin value. However, the primary purpose of the wallet is to store the actor's private key.

Each bitcoin transaction has two scripts associated with it, written in the Bitcoin scripting language. These are: (i) "scriptPubKey" (used to spend bitcoins by signing them to another actor); and (ii) "scriptSig" (used to prove that an actor is allowed

to spend bitcoins). These scripts must both be successfully run by the Miner that is validating the transactions, with each script needing to meet the checks to prove that the transaction is valid. Data pushed to the Miner, also includes the sender's signature and public key. The Miner then derives the sender's bitcoin address using the sender's public key (both of which can only have been created with the sender's private key). If the resultant sender address matches the actual sender address for the transaction, and the signature packaged within the transaction is confirmed, the transaction is considered valid. The Miner will then add the transaction to the queue of transactions, and when the next block is mined (discovered) the transaction will be added into the new block. To get the transaction mechanism between Alice and Bob started, Alice first requires bitcoins and an address. Let us say that Alice had previously created a private key and a public key from which her bitcoin address was generated. Alice had also previously received bitcoins from Jim (20 bitcoins), Kate (10 bitcoins) and Linda (35 bitcoins) to her bitcoin address: a total of 65 bitcoins. There are two key details in respect to the transactions from Jim, Kate and Linda to Alice:

1. The blockchain transaction mechanism requires Jim, Kate and Linda to sign each of their transactions to Alice's bitcoin address with their own private key related to the bitcoin address that the bitcoins are being sent from. Anyone with the public key of Jim, Kate or Linda's bitcoin addresses can validate that they actually had entitlement to the bitcoins contained in the "send" transaction to Alice in the first place.
2. As part of the Bitcoin transaction protocol, Alice's bitcoin address is included in the transaction. This means that only the person with the private key to that address (Alice!) will be able to spend the bitcoins (by re-signing a new transaction to a new bitcoin address).

These precursory transactions are essential for explaining the chain of digital signatures,

as Alice now has digital signatures from Jim, Kate and Linda signed to her bitcoin address that represent the bitcoin inputs for her transaction to Bob.

A.2.2 Inputs

Transaction inputs are created from the hash digest of the transactions from Jim, Kate and Linda that have the bitcoin values that Alice wants to send. For example, if Alice wants to send Bob 50 bitcoins, she needs to have sufficient "bitcoins" in her "Bitcoin wallet". These are the transactions in the public ledger that have been signed over to Alice's bitcoin address, and each transaction holds the respective public keys for Jim, Kate and Linda in the output scripts, as well as the bitcoin values. Since Alice has received 20 bitcoins from Jim, 10 from Kate and 35 from Linda, there are 3 input addresses required for Alice to send 50 bitcoins to Bob.

These input transactions, originally from Jim, Kate and Linda, are individually hashed to create a series of digests that relates to the addresses that Alice received the bitcoins from. Each digest is then included as an input on the new transaction. This allows anyone to verify chain of ownership of the previous transactions, because the public key of each source address is stored in the transaction script, and is used by the Miner to validate the input transactions from Jim, Kate and Linda.

A.2.3 Outputs

The output transaction requires a list of destination addresses and the bitcoin values. Any balance from the input quantity of coins must be returned to a sender address or as a Miner fee, as outlined in Chapter 2.5.5 on page 73. Hence, for Alice to send 50 bitcoins to Bob, the outputs would include the destination address for Bob with 50 bitcoins, and an output back to Alice's address of 14 leftover bitcoins (for example) and 1 bitcoin in mining fees to pay to the Miner (for example), ensuring a total of 65

bitcoins are spent in the output of the transaction. Finally, Alice digitally signs the entire transaction data of inputs and outputs with her private key relating to the address that is sending the bitcoins to to Bob, and includes the digital signature in the transaction itself (which is ultimately immutably stored on the blockchain).

It should be noted that Alice's private keys are never exposed throughout the transaction process. Otherwise, Alice's private key information would be exposed either in transit between Alice and the Miner, exposed at the Miner during processing, or exposed on the public blockchain itself.

A.3 Sending the Transaction

Sending the transaction to the Miners in a cryptocurrency peer-to-peer network is a transport mechanism rather than a data mechanism. The processes that discover the P2P network, and subsequently the actual IP (Internet Protocol) addresses of the Miners, does not change the data. All nodes in the Bitcoin ecosystem listen to broadcasts from clients or other Miners and decide if they need to act on the message contained within the broadcast.

Beyond Bitcoin, other cryptocurrencies utilise a similar mechanism to connect clients and Miners, although there are some varying architectures in respect to Miner status and discovery. Cryptocurrencies such as Ripple[1] and Dash[2] use a "Master-node" concept to establish the authoritative level of the Miners and level of trust. Ripple utilises a closed Master-node system that Ripple explicitly approves, whilst Dash Master-nodes literally buy in to the Dash network to fund being a Master-node.

[1] https://www.ripple.com
[2] https://www.dash.org