VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering
and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

Anonymously Establishing Digital Provenance in Reseller Chains

by

Benjamin Philip Palmer

A thesis

submitted to the Victoria University of Wellington
in fulfilment of the requirements
for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2012

Abstract

An increasing number of products are exclusively digital items, such as media files, licenses, services, or subscriptions. In many cases customers do not purchase these items directly from the originator of the product but through a reseller instead. Examples of some well known resellers include GoDaddy, the iTunes music store, and Amazon.

This thesis considers the concept of provenance of digital items in reseller chains. Provenance is defined as the origin and ownership history of an item. In the context of digital items, the origin of the item refers to the supplier that created it and the ownership history establishes a chain of ownership from the supplier to the customer. While customers and suppliers are concerned with the provenance of the digital items, resellers will not want the details of the transactions they have taken part in made public. Resellers will require the provenance information to be anonymous and unlinkable to prevent third parties building up large amounts of information on the transactions of resellers. This thesis develops security mechanisms that provide customers and suppliers with assurances about the provenance of a digital item, even when the reseller is untrusted, while providing anonymity and unlinkability for resellers .

The main contribution of this thesis is the design, development, and analysis of the tagged transaction protocol. A formal description of the problem and the security properties for anonymously providing provenance for digital items in reseller chains are defined. A thorough security analysis using proofs by contradiction shows the protocol fulfils the security requirements. This security analysis is supported by modelling the protocol and security requirements using Communicating Sequential Processes (CSP) and the Failures Divergences Refinement (FDR) model checker. An extended version of the tagged transaction protocol is also presented that provides revocable anonymity for resellers that try to conduct a cloning attack on the protocol. As well as an analysis of the security of the tagged transaction protocol, a performance analysis is conducted providing complexity results as well as empirical results from an implementation of the protocol.

Acknowledgements

I would like to thank my supervisors Kris Bubendorfer and Ian Welch for their help throughout this thesis. I would also like to thank my examiners for the useful feedback and discussion. Last but not least I would like to thank my family and friends for their support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Provenance is defined as the origin and ownership history for an item. In art history the provenance information for a painting is used to help decide whether the piece is real or a forgery. Locally, the notion of provenance attracted attention when in 1985 Karl Sim was arrested for art forgery after copying and selling art works by famous New Zealand painters such as Charles F. Goldie, Frances Hodgkins, Rita Angus and Colin McCahon. By signing the artist's name on the painting he successfully passed himself off as the artist and sold the art works at auction. The number of forgotten masterpieces appearing in the same place with no provenance information alerted authorities to the forgeries.

The concept of provenance can be applied to data as well as art work. Data provenance provides provenance meta data for scientific workflows [34, 87, 92], databases [3, 91], geographic information systems (GIS) [53], and web services [39, 89]. This thesis examines the concept of provenance for digital items such as digital media or more abstract products such as an access 'right', a license, a service, or a subscription. In many cases customers purchase digital items through resellers instead of directly from the suppliers of the items. Some well known resellers that exist only as on-line traders include Amazon, iTunes, and domain name resellers, such as GoDaddy.

A simple approach to achieving digital provenance in reseller chains is to introduce a license server that acts as a trusted third party. This license server can check at every step in the transaction that the item is legitimate and that it has not been sold to multiple customers. A downside of this approach is the potential for privacy breaches because the license server has control over a large amount of data both on the details of transactions conducted and the identities

of the parties involved. A better option is to provide verification of the actions of any third party in the protocol without reducing privacy.

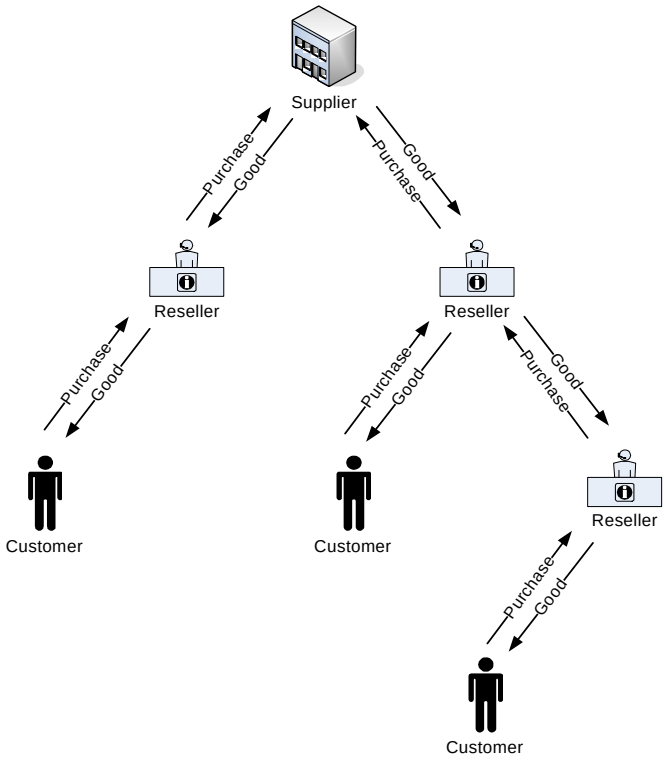

Figure 1.1: The Reseller Model

The basic reseller model is shown in Figure 1.1. This work refers to each individual party in the model as a link and the entire path from the supplier to a customer as a chain. There are three parties involved:

- The Supplier: Suppliers are the original creators or holders of the rights for an item.
- The Reseller: The reseller or middleman has a set of customers who purchase goods from them as well as a set of suppliers that the reseller purchases the goods from. There may be multiple resellers between the supplier and the customer.
- The Customer: Any party that is interested in purchasing an item produced by a supplier and sold by resellers.

This thesis defines the term *secure provenance* where correct provenance information is supplied in the presence of active adversaries without reducing

privacy for participants in the protocol. Secure provenance provides customers with confidence in the origin and ownership history for a digital item *even when the reseller they are dealing with is untrusted*.

The approach taken in this thesis differs from the use of digital certificates and Digital Rights Management (DRM) in crucial ways. Most Internet resellers use a digital certificate to prove their identity and to provide information on their physical location and contact details. Digital certificates do not provide mechanisms to establish the provenance of items as they are intended as a way of establishing identity.

DRM is an aggregation of security technologies to allow content owners to maintain persistent ownership and control of their content. A DRM system usually wraps the content in a secure container that prevents unauthorised users from accessing the content. Most DRM systems assume the reseller is trusted but this work assumes an untrusted reseller and does not provide methods to establish the provenance of an item.

When providing provenance information, there are many ways an adversary could create false provenance information or incorrectly modify existing provenance information. In this thesis, I consider attacks that fit into one of the following four categories:

1. Spoofing. The adversary claims to be the supplier or tries to subvert the protocol to make it appear that they are the supplier. In this way the adversary may be able to take payment for an item without ever paying the supplier while the customer believes they have correct provenance information.

2. Counterfeiting. The adversary sells the customer an item but never buys it from the supplier. There are several ways an adversary could counterfeit provenance information for an item including:

    - Fabrication. The adversary fabricates provenance information for an item from scratch (or having seen the structure of other provenance information over the network).
    - Cloning. The adversary sells an item they have purchased from the supplier to multiple customers. The adversary will have legitimately purchased the item initially but is trying to sell multiple copies when it has only purchased one.

- Network Sniffing. The adversary sees legitimate provenance information for another reseller or customer being sent over the network, copies it, and sends it to a customer.

3. Identity Revelation. An adversary learns the identity of a participant in the protocol that is not its neighbour in the chain. The participants in the protocol will know the identity of their neighbours in the chain, for example, a customer will know the identity of the reseller it is purchasing an item from.

4. Linkability. An adversary links together the actions of a party, other than the supplier, in two separate runs of the protocol. The adversary will not need to discover the identity of the participant, just link their actions together. The supplier is excluded from these attacks as the supplier is always the participant that first generates the provenance information for the item, although they may be anonymous.

1.1 Use Cases

To motivate the requirements and provide context for providing provenance for digital items in reseller chains three example use cases are considered.

Use Case 1 - Online Music Stores

Online music stores are a development from the traditional bricks and mortar music store. A bricks and mortar music store sells tapes, CDs, and records to customers as physical items. Once a customer has purchased an item, they can play it in any location that the license applies (licenses often do not apply for public performance of the work or use in a film). In many cases, when the customer has finished with the item, they can sell it back to the music store who sells it as a second hand item to other customers. Alternately, the customer can sell the physical item directly to another customer. When a physical music store does not have an item, they have to order it in and it is sent to them through the postal service.

An online music store sells customers a digital copy of the physical items sold by a physical music store. These digital items contain the music tracks, and a digital version of any accompanying booklets or information. Unlike a bricks

and mortar music store, an online store cannot 'sell out' of a digital item as the customer is sent a copy of an item. The online music store can keep making as many copies of the digital item as they need. To provide the most efficient service to customers, the online music store may want to pre-purchase a bulk set of licenses for the item in one go. These can then be sold to the customers with no interaction from the supplier.

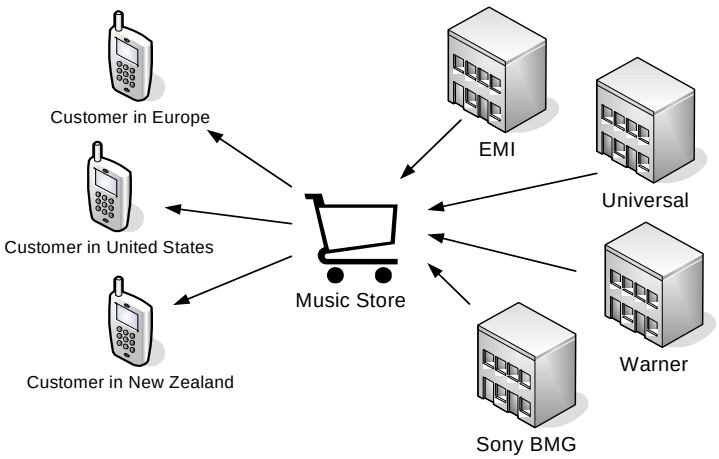

Figure 1.2: An Online Music Store

The Apple iTunes music store has become the largest music retailer in the United States [4] with sales of over five billion songs [5]. While customers may have a strong incentive to trust such a large business, there are many smaller music stores that may not have the same amount of pre-existing trust. A customer who finds a track cheaper at another store requires mechanisms to verify the provenance of the item.

Digital rights management (DRM) was used by the iTunes music store to limit the number of times their customers can make a compact disc copy of purchased songs and the number of computers the customers can access purchased songs from. However, DRM does not provide methods to establish the provenance of an item.

Use Case 2 - eBook Resellers

A bricks and mortar book store sells books and magazines to customers. Similar to the bricks and mortar music store, a book store must order in stock to sell to customers and can possibly sell out of a title or have to order in extra stock. A

large number of second hand book stores exist where customers can take their unwanted books to resell to other customers.

In the digital model, publishers produce eBooks which are then sold either direct to customers or through a reseller as shown in Figure 1.3. There may be several resellers in between the customer and the publisher. An eBook reseller takes books that they have either self published or brought from a publisher and sells them to a set of customers. Examples of eBook resellers include Amazon, Barnes and Noble, and O'Reilly. Some eBooks are protected by DRM that limits printing, copying, and redistribution. Increasingly eBooks are being sold to customers using eBook readers such as the Amazon Kindle, smart phones, and the Apple iPad. As many eBook readers have the capability to remotely disable access to an eBook, the customer has a strong interest in verifying the provenance for the eBooks they purchase. As many of these devices are low power devices, any protocol for establishing the provenance of eBooks must be as efficient as possible.

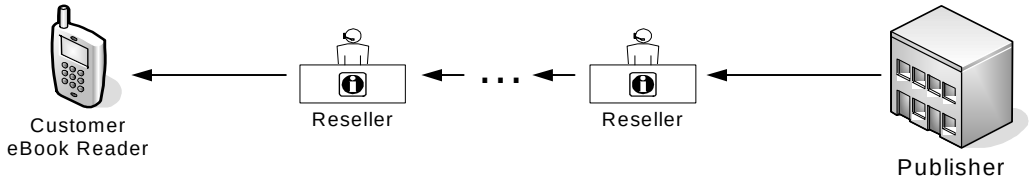

Figure 1.3: An eBook Reseller Chain

Combining the properties of the bricks and mortar book store and the online eBook resellers would give several advantages. When selling eBooks, the reseller cannot run out of stock as a new copy of the item can be copied. When using an online reseller, customers can purchase eBooks 24 hours a day 7 days a week rather than having to wait for the physical book store to be open. Customers would benefit if they could resell eBooks they had finished reading in the same way as they can now with a physical book.

Use Case 3 - Domain Name Registrars

Figure 1.4 shows the parties in the domain name marketplace. Domain name registrars are Internet Corporation for Assigned Names and Numbers (ICANN) registered companies or organisations that sell top level domain names either to

customers or to domain name resellers. A domain name reseller sells domain names to either customers or other resellers. There may be several domain name resellers between the customer and the domain name registrar. A customer may not know how many resellers are between it and the registrar, or what registrar the reseller is using. The reseller may want to keep the identity of the registrar it uses private to prevent a customer from going straight to the registrar to get a better deal. Customers can switch their domain names between different resellers or registrars. Some examples of domain registrars are GoDaddy, OnlineNIC, and Domainz. Domain name resellers include ImHosted.com and ResellerClub.com.

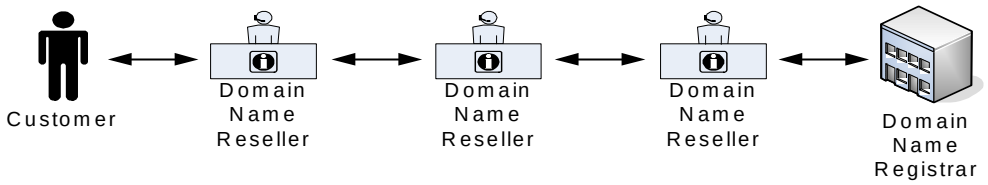

Figure 1.4: Domain Name Resellers

Domain name resellers have not always acted honestly. Registerfly was a domain name registrar accredited by ICANN with over 200,000 customers [41]. Customer complaints with Registerfly included allegations of fraud, incorrectly altering "whois" data, and suspension of accounts in retaliation for complaints about over charging. ICANN eventually terminated Registerfly's accreditation on March 31st 2007 [49].

1.2 Requirements

Based on the use cases presented, I now list the requirements for a protocol for anonymously establishing provenance in reseller chains.

- R1: Establish Provenance of Digital Items: The central requirement must be to enable customers to establish the provenance of digital items they are purchasing. The customer does not need to know which suppliers or resellers are in the chain, but they must be confident in the provenance of the digital items they purchase in the presence of malicious resellers.

- R2: Support Multiple Resellers between the Supplier and the Customer: There may be several resellers between the supplier and the customer. The

protocol must be able to establish the provenance of an item when there are a dynamic number of resellers between the supplier and the customer.

- R3: Support Customer Reselling of Items: Customers should be able to resell a digital item they do not require. In this scenario, the customer would be acting as a reseller and on selling to another customer.
- R4: Provide Anonymity: Suppliers and resellers in the reseller chain need to be anonymous to prevent a customer or reseller from trying to skip the middleman and contacting a reseller or supplier who is further up the chain to obtain a better deal.
- R5: Unlinkability of Actions by Resellers: To prevent a third party from building up detailed records of past transactions of a particular reseller or customer, the transactions of a reseller or customer need to be unlinkable between separate transactions.
- R6: Efficient for Low Powered Devices: As some customers will be using low powered devices, the protocol needs to be efficient for customers.
- R7: Agnostic to Item Delivery Method: Many methods are used to deliver digital items from direct downloads to peer to peer systems. The protocol needs to establish the provenance of a digital item without relying on a specific delivery method.
- R8: Allow Bulk Buying of Items: A reseller may wish to bulk purchase a set of items so they do not have to keep interacting with the supplier to sell an item to a reseller. If a protocol supports bulk buying then the supplier can be offline when the customer is purchasing items from a reseller.

1.3 Thesis Summary

This thesis examines the problem of providing secure provenance for digital items in reseller chains. Current schemes that can provide provenance in reseller chains are presented. These schemes are evaluated and compared based on the requirements presented in this chapter. This evaluation concludes that none of the current systems are able to fulfil all the requirements.

Using terms for expressing provenance models, a model of the provenance information for digital items in reseller chains is then constructed. This model

shows the conflict between providing complete provenance information and privacy. To provide secure provenance a formal definition of secure provenance is required to evaluate the protocols provided in this thesis and provide convincing arguments that these protocols are secure. The formal secure provenance definition is built on the formal definitions of spoofing, fabrication, cloning, network sniffing, identity revelation, and linkability attacks.

The tagged transaction protocol is developed to provide secure provenance for digital items in reseller chains. The tagged transaction protocol provides confidence in provenance information in the presence of active adversaries, provides anonymity and unlinkability for participants, supports customer reselling of items, and offline suppliers after the item has initially been sold. The tagged transaction protocol does not provide enforcement of the terms of licenses. The tagged transaction protocol uses a third party called the Tag Generation Centre (TGC) and data structures called tags to provide provenance information. A thorough security analysis of the tagged transaction is presented showing that the tagged transaction protocol provides secure provenance. The tagged transaction protocol is analysed using the Failures Divergences Refinement (FDR) state based model checker to show that it provides security against spoofing, fabrication, cloning, and network sniffing attacks.

This thesis also develops a second protocol to provide secure provenance called the extended tagged transaction protocol. The extended tagged transaction protocol has the same features as the tagged transaction protocol but also provides revocable anonymity for resellers who try to perform cloning attacks. The extended tagged transaction protocol uses ideas from digital cash where customers and reseller withdraw coins from the TGC and spend these coins when they regenerate tags. If a customer or reseller clones a tag, the digital coin can be used to revoke the identity of the customer or reseller but does not reveal identity or linkability information when the coin is used once. The extended tagged transaction protocol has had a thorough security analysis and has also been analysed using the FDR model checker.

The use of the TGC introduces a trusted third party. Two mechanisms have been developed to verify the actions of the TGC: (1) The use of a public bulletin board where the TGC publishes all its actions; and, (2) Distribution of trust over multiple TGC replicas where, as long as a threshold value of the TGCs are honest, customers, resellers, and suppliers can have confidence in the provenance information provided. Mechanisms to provide an anonymous communication

channel are also considered to provide anonymity for customers, resellers, and suppliers.

The performance of the tagged transaction protocol and the extended tagged transaction protocol are examined in terms of computational complexity, communication complexity, and experimental performance. Both the tagged transaction protocol and the extended tagged transaction protocol have been implemented in Java. This implementation has been used to perform experiments to examine the effects of cryptographic key size, number of resellers, the threshold number of TGCs when using a distributed TGC, and the anonymous communication channel on the performance of the protocols.

The ideas and mechanisms developed in the tagged transaction protocol to provide secure provenance in reseller chains for digital items have been applied to providing secure provenance in web services. The web services model has several differences to the digital item model. Web services provide a service that is on going as opposed to the one off nature of transactions for digital items. Web services also do not have the same privacy requirements as reseller transactions. A solution is presented and analysed in terms of the security properties of the protocol. A definition and discussion of exclusion attacks where a service provider does not provide all the inputs it uses to provide the service in the provenance information is also presented.

1.4 Contributions

The main contributions of this thesis are:

1. Formalising the requirements and algorithms for a protocol for anonymously providing provenance in reseller chains. Based on the possible attacks, this results in a formal definition of secure provenance.

2. Development of the tagged transaction protocol and the extended tagged transaction protocol. Both protocols use tags and a third party called the Tag Generation Centre (TGC) to anonymously provide provenance in reseller chains. This contribution has been published as a paper "A Protocol for Anonymously Establishing Digital Provenance in Reseller Chains" in Financial Cryptography 2011 [68] and has been patented as "NZ 585382: Method for Providing Anonymous Authentication".

3. Security analysis of the tagged transaction protocol using arguments by contradiction and model checking.

4. Implementation and performance measurements of the tagged transaction protocol showing the effects of key size, number of resellers, the choice of distribution, and anonymous communication channel used.

5. Application of the ideas from the tagged transaction protocol to provide secure provenance in web services. This includes the definition and discussion of methods to prevent exclusion attacks. This work has been published as a paper "Verifying Digital Provenance in Web Services" in the Privacy and Provenance in the Cloud 2011 workshop [69].

1.5 Thesis Organisation

The rest of this thesis is organised as follows: Chapter 2 presents related work, Chapter 3 provides the formal definition of secure provenance, Chapter 4 presents the tagged transaction protocol and the security analysis of the tagged transaction protocol, Chapter 5 presents the extended tagged transaction protocol and the security analysis of the extended tagged transaction protocol, Chapter 6 presents ways to provide an anonymous communication channel and mechanisms to verify the actions of the TGC, Chapter 7 presents the details of the implementation of the tagged transaction protocol and details of the performance results, Chapter 8 presents the application of the ideas from the tagged transaction protocol to providing provenance for web services, and Chapter 9 presents conclusions and future work.

Chapter 2

Related Work

Data provenance is a wide area of research that includes work on provenance in Geographic Information Systems (GIS), scientific workflows, databases, and attack tracing. Data provenance records the origin and modification history of data. One of the earliest works in data provenance was by Lanter for GIS [53] in 1991. In this work, provenance of spatial GIS data was represented as a bi-directional graph where nodes represent a map layer and edges represent a transformation from one map layer to another. A frame object was also associated with each map layer that recorded additional provenance information about the layer.

Scientific workflows provide means for scientists to perform advanced scientific tasks in a collaboratory environment. A collaborative environment is realised by the use of publicly accessible data sources, Grid middleware, and web services. These environments can span multiple organisations or groups of researchers. Provenance information allows researchers to reproduce results or detect which data sets an erroneous result has affected. There have been several provenance systems designed to work with scientific workflows including Chimera [34] and myGrid [87, 92].

Chimera provides provenance information for data intensive applications. Provenance is recorded as data derivation steps from one data set to another. On demand regeneration of data is possible using the provenance information. Users can query the Chimera provenance system for a directed acyclic graph that represents the tasks executed to create a specified data product. myGrid executes workflows and automatically generates provenance records based on the data set used, execution times, and workflow. Every entry in the experimental

components store has a provenance object associated with it. Scientists can make annotations to provenance logs with information such as the hypothesis of the experiment and any thoughts and opinions.

Tioga provides an environment to graphically compose data and programs in a database environment [3, 91]. The provenance information is represented as inverse functions. For a function with input I that produces I^1 the inverse function takes I^1 and produces I. For functions that do not have a direct inverse Tioga introduces the concept of a weak inverse function.

Provenance information has also been used to detect the source of attacks in Grid nodes [38]. The Grid node keeps a record of the file and process identifiers when a file is read or written and when a process is created. When a Grid node detects an attack, the provenance graph for the process or file is then extracted assisting in the identification of the source of the attack.

The Open Provenance Model (OPM) aims to design a shared provenance model for all provenance systems [59]. The use of a shared provenance model allows the interchange of provenance information between different provenance systems. The OPM does not provide details on how provenance systems should be implemented. OPM defines three main entities. An artifact is piece of state which could be some data or a physical object. A process is a series of actions that act on an artifact to produce a new artifact. An agent is an entity that controls a process. The OPM then provides a series of terms that define causal relationships between artifacts, processes, and agents. Provenance information is then represented as a directed graph where artifacts, processes, and agents are nodes and causal relationships are edges.

The open provenance model is applied to distributed systems in work by Groth et al [40]. Messages sent between parties are modelled as artifacts sent between processes. The data items in the message are attributes that are extracted from received messages. They use a D-Profile to represent distributed systems in the OPM that provides a more compact representation than the fundamental OPM constructs but can be expanded into an OPM graph.

A high-level view of the security issues when providing provenance information in a distributed environment is presented by Tan et al [89]. They discuss the issues of access control for provenance information, trust issues with provenance providers and stores, accountability of provenance information, and long term storage of provenance information. In the context of this work the issues of trust for provenance providers and stores and accountability of provenance

information are both of interest. Accountability is normally addressed through digital signatures. Another issue is the trustworthiness of the provenance store. This work uses mechanisms to provide verification of the third parties acting as the provenance stores. Long term storage of provenance information is an interesting area with important problems as the amount of provenance information gathered will keep increasing. Other issues include the format of data that may change and the updating of keys that are used to sign or encrypt provenance information. Long term storage of provenance information is out of the scope of this work.

Access control for provenance stores is addressed in a number of works [12, 22, 89]. Provenance information may contain sensitive information. Access control methods are considered that provide access in varying levels of granularity. Provenance information differs from traditional data in that it is about relationships and is not a single piece of data. The data resulting from a process and the provenance information for the process may also have different privacy requirements. Access control for provenance systems is out of the scope of this work.

2.1 Specific Systems

This section compares protocols that can be used to provide provenance for digital items in reseller chains. Two Digital Rights Management (DRM) systems are included for completeness. These protocols are then compared based on the requirements listed in Section 1.2. After the comparison, a protocol that provides provenance in web services in the presence of active adversaries is examined and compared to the approach in this thesis.

2.1.1 Apple Fairplay System

In the Apple Fairplay system, Advanced Audio Coding (AAC) format audio streams are encrypted using a master key. This master key is also encrypted and stored with the encrypted song. The key required to decrypt the master key is called a user key. When a user purchases a song, a user key is generated for them and the song encrypted with the master key and the master key encrypted with the user key are sent to the iTunes music player. The iTunes music player stores the user key and it is also stored on the iTunes server. Using the user key, the

iTunes music player can decrypt the master key for the song and then play the encrypted audio stream. When a user registers a new computer with the iTunes store, the iTunes store sends the user generated keys corresponding to that user. The Apple Fairplay system provides no way for the user to check the provenance of the item they are receiving, they have to trust that the iTunes music store is sending them legitimate content that it has purchased from the correct supplier. There is no mechanism for customers to resell content.

2.1.2 Open Mobile Alliance DRM

The Open Mobile Alliance have a DRM system that makes use of a trusted DRM Agent and a third party known as the Rights Issuer (RI) [66]. Content that has been protected is encrypted using a key that is delivered in a rights object. When a DRM Agent downloads content, it contacts the RI responsible for that content. The rights object for that content is then sent to the DRM Agent encrypted using a key bound to the DRM Agent. The DRM Agent can then decrypt the rights object to access the content. The Open Mobile Alliance DRM system depends on a trusted DRM Agent and a trusted RI. It does not provide any mechanisms to verify the provenance of the media. There is no mechanism for customers to resell content.

2.1.3 Superdistribution

The concept of superdistribution was invented in 1983 by Ryoichi Mori who originally termed it the Software Service System as it was envisaged as a way of delivering software similar to the way the water system delivered water [60]. A superdistribution model involves the buyers of digital goods in the distribution process [82]. A superdistribution model involves a content distribution scheme and a remuneration scheme. The content distribution scheme can be any technology such as peer to peer (p2p) transfers, http transfers, or blue tooth. Each digital good distributed over the content distribution scheme contains license information that describes how the content may be used, by whom, and what remuneration must be paid and how. The remuneration scheme provides mechanisms for the buyer of a digital good to make a payment for the redistribution of a digital good and for the payment to be correctly split amongst the parties specified in the license. The mechanisms to anonymously

establish provenance for digital items in reseller chains can be applied to the superdistribution model.

2.1.4 The Paradiso System

The Paradiso system lets customers purchase not only songs and videos from suppliers but also reseller rights so they can sell a certain number of the purchased songs and videos to other customers [61, 62]. This system enables any customer to become a reseller taking advantage of word of mouth and taste-orientated marketing. Once a customer has bought the rights to redistribute a certain number of copies from the supplier, they can distribute these without having to contact the supplier.

The Paradiso system is designed to be run on media devices, like the iPod or Zune, with a trusted computing module (TCM) to enforce the contracts between supplier and customer. Each player has a private key that is stored in the secure hardware of the TCM that is loaded on by the manufacturer. All private key operations are performed in the secure hardware. All songs are stored encrypted with an individual symmetric key which is stored in the secure hardware.

To purchase a song and the rights to resell it N times, the player sends a request to the content provider or supplier containing the request for the purchase as well as the player's public key. Once the payment from the customer has cleared, the provider encrypts the content with a newly generated symmetric AES key. The content provider then sends the encrypted content along with the symmetric AES key and any reselling rights encrypted with the player's public key. The reselling and license information are signed using the supplier's secret key. Once the device receives the license information, it checks that it has been signed by a valid reseller, and then stores the license information (still encrypted with the player's public key) to insecure memory.

The Paradiso system can be used to provide secure provenance information provided a valid TCM is present in the device. While currently the Paradiso system does not provide provenance information it could be included with the license information. The Paradiso system does not provide any anonymity or unlinkability for participants.

While the Paradiso system has many attractive properties, its security rests on a valid TCM. Providing every device (computer, media player, netbook) has a TCM, Paradiso will effectively allow reselling of media without the input of

the original content provider. The Paradiso system can be used to provide secure provenance information provided a valid TCM is present in the device.

2.1.5 The Potato System

Fraunhofer and a spin-off company 4FriendsOnly have developed a music redistribution protocol called the potato system [1, 75]. The central idea of the potato system is to reward users with a percentage of the song payment for redistributing content. The users of the potato system do not just pay for content, they also pay for a re-distribution license. They can then re-distribute the song as many times as they like and will be rewarded a set proportion of the purchase cost each time the song is purchased by a user they have distributed the song to. Content in the potato system is transferred over peer to peer (P2P) networks, although to transfer or purchase content requires interaction with the centralised potato system web interface.

Nutzel et al describe using a signed media format for the potato system [64]. In a signed media format (SMF) the media content is symmetrically encrypted using the advanced encryption standard (AES). The key for the AES encryption is then encrypted with the private key of the last buyer. A SMF consists of the encrypted media, the encrypted AES key, the public certificate of the last buyer, the license from the accounting server, and a signature of the license by the accounting server. The license includes the name of the content owner, a hash of the content, the name of the last buyer, the price model used for this media file, the link to buy the file, and other information. The accounting server in this case is the potato system. By using a SMF, any player that wants to play the media has to decrypt the content using the public certificate of the last buyer and can pop-up a window asking if the user wants to purchase a re-distribution license if the last buyer was not the current user.

While the potato system allows for redistribution of content using P2P networks, interaction with a trusted third party in the form of the potato system web interface is required to purchase songs. The potato system is an interesting example of the use of financial rewards as opposed to strong cryptographic methods. The potato system could provide provenance information along with the chain of ownership and license information for the song provided when a song is purchased or listened to. This provenance information would be created and updated by the centralised potato system web interface. Like the

rest of the potato system the provenance information would not be protected by cryptographic methods but using financial rewards.

2.1.6 Contracts for a Distribution Chain

Durfee et al have designed and implemented a protocol using contracts for distribution chain security [31]. A contract is a sequence of string tokens written in a contract language that lists the rights granted to the holder of the contract. A contract is created for a digital work by the content provider or author. A contract may include terms that detail obligations to pay royalties to the content provider, or an expiration date for the right to use the digital work. A contract can be obfuscated to hide sensitive values in the contract by using commitment values.

Once a contract has been created by a content provider for a reseller it is sent to a contract certifier who checks the contract is well formed and signs it with a digital signature. The contract certifier is a third party that certifies any newly created contracts to ensure that a new contract is faithful to any existing contracts on the digital work. Contracts sent to the contract certifier can be obfuscated to hide any sensitive data and shown to be valid using zero knowledge proofs that a committed value lies in a certain range.

When a reseller has a digital work and certified contract from the content provider, they can then create a new contract and on sell the digital work to a customer. A new contract is created by sending the contract certifier an obfuscated version of the new contract along with the signed obfuscated original contract. The contract certifier then checks that the new contract is faithful to the old contract using zero knowledge proofs for proving relations on committed numbers [17] to verify obfuscated fields. The contract certifier then signs the new obfuscated contract and sends it to the reseller.

When a customer acquires the digital work from the reseller, the reseller also sends them the signed obfuscated contract from the contract certifier as well as an unobfuscated contract. The customer then checks that the new contract is the same as the new obfuscated contract and that the new obfuscated contract has been signed by a contract certifier. The customer will only accept digital works that are accompanied by a valid contract that has been signed by a contract certifier.

The authors also suggest a similar protocol that does not involve a contract certifier. The contracts are chained together and the zero knowledge proofs that

the older contracts are faithful to the new contracts are attached to the contract. This increases the size of the contracts but has the advantage of removing the third party.

This work is set in the domain of reseller transactions. Rather than looking at anonymously establishing the provenance of the digital items, they concentrate on establishing the correctness of the license while hiding sensitive data. This protocol could be used in conjunction with a protocol for providing provenance for digital items to provide a method for resellers to alter the terms of the license but restricting the modifications to only allow valid modifications for that license.

2.1.7 A Decentralised and Secure Electronic Marketplace

Serban et al introduce the concept of a decentralised electronic marketplace (DEM) where transactions are subject to a set of trading rules [85]. Trading rules define the DEM and may be different for different marketplaces. The trading rules are implemented using a mechanism called Law Governed Interaction (LGI). LGI is a mode of interaction that allows an open group of distributed heterogeneous agents to interact with each other with confidence that the agents are all following the law of the marketplace. In LGI, a law is formulated using an event-condition-action pattern. Apart from the agents taking part in transactions in the marketplace, there are also a set of trusted controllers that enforce the law of the marketplace. Every agent is assigned a controller, and all messages for the marketplace are sent through the agent's controller.

The controllers in LGI store information on the local state of the agent they are monitoring. This enables a controller to pick up on actions that are not allowed according to the trading laws governing the marketplace. For example, if a reseller tried to sell a license for digital media they did not own or tried to sell a license twice to different customers the controller would detect this and prevent it. The controllers are assumed to be trusted if they have a certificate signed by the certification authority for this marketplace which is specified in the trading laws. Several agents in the DEM can share the same controller.

For agents in a DEM to have confidence that other agents in the DEM are obeying the trading laws, the controllers need to be trusted. The authors suggest the controllers need to be provided as a public utility by a large financial or governmental institution that can act as a trusted third party and has no interest in the computing activities regulated by its controllers. When reselling

digital content in DEM the trusted controllers could be used to create and update provenance information. Provided the distributed controllers are acting correctly DEM can provide correct provenance information for digital items. The controllers build up a large amount of information on the transactions being done and the identities of the participants in the transactions. If a single party is in charge of the controllers, this single party gains all the information on the transactions happening in the marketplace.

2.1.8 IEEE P1817 Working Group

The IEEE P1817 working group has produced an initial description of a system to allow consumers perpetual ownership and unrestricted use of copyrighted materials [67]. The only limitation put on the use of copyrighted material is that a key is required to use a product. The idea of the working group is to allow users to use, share, lend, give, and resell, but not copy, digital materials. While the P1817 standard provides options for customers to resell content it relies on a trusted player to store cryptographic keys and not allow them to be copied. The working group is still in its early stages and it will be interesting to see how this develops in the future.

2.1.9 Anonymous Credentials

The Idemix system [13] developed by Camenisch et al is an implementation of an anonymous credential system [15]. A credential system has users and organisations. Credentials are issued by organisations to users. Users then demonstrate possession of these credentials to other organisations. When using anonymous credentials, users are known to organisations by pseudonyms and a series of independent transactions by a user are unlinkable. A user has different pseudonyms with different organisations. Credentials can be multiple use or one-show. In this thesis I concentrate on one-show credentials as these can be used to implement a protocol for anonymously establishing digital provenance in reseller chains. If a user can prove possession of a one-show credential for an item, then the provenance of the item can be established.

In anonymous credentials, each organisation has a private and public key. Each user has a private key. A pseudonym for a user is a name by which the user is known to an organisation and is formed by a user generated section and

an organisation generated section. A pseudonym is tagged with a validating tag that is statistically independent of the user's private key. A credential is a tuple that cannot be forged for a correctly formed validating tag. To show possession of a credential a user takes part in a protocol with the organisation using zero knowledge proofs of knowledge.

Extensions to the basic anonymous credential protocol support one-show credentials with optional local revocability (revealed to the issuing organisation) and global revocability (revealed to all parties) when a user tries to use a one-show credential multiple times. All-or-nothing non transferability prevents credential pooling where, if a user lends a credential to another user, all the details about the lending user are revealed to the other user allowing identity theft.

One-show anonymous credentials can provide anonymous digital provenance in reseller chains. When using one-show credentials, the organisation that generated the original tag would have to be online to verify and generate a new anonymous credential. Anonymous credentials do not support supplier anonymity as only the users are anonymous and not the organisations.

2.1.10 Comparison and Discussion

Table 2.1 shows a comparison of related work. The first column of the table lists the name of the protocol. The second column lists the trust model or what trusted party and/or cryptographic techniques the protocol uses. The third column lists whether the protocol can be adjusted to provide secure provenance information. The fourth column lists whether the protocol provides verification of any third parties, the fifth whether the protocol supports offline suppliers, and the sixth column indicates if the protocol supports anonymous suppliers. The final column indicates whether the protocol supports customer reselling of content.

While the protocols discussed in this chapter achieve some of the goals of a reseller verification protocol all have disadvantages. The Apple Fairplay system relies on a trusted reseller and does not provide any methods to establish the provenance of digital items. The Open Mobile Alliance DRM relies on a trusted player. Neither of the DRM system provide the option for the customer to resell content.

The Paradiso system is heavily reliant on a TCM and does not provide the option for an anonymous supplier. However, the Paradiso system can provide secure provenance information and provides support for customer reselling of

Protocol	Trust Model	Can Provide Provenance	3rd Party Verifiable	Offline Supplier	Anonymous Supplier	Customer Reselling
Apple Fairplay	Trusted Reseller and Encrypted Media			✓	✓	
Open Mobile Alliance DRM	Trusted Player and and Rights Issuer			✓	✓	✓
Paradiso System [62, 61]	Trusted Computing Module	✓		✓		✓
Potato System [1, 75]	Financial Incentives	✓		✓		✓
IEEE P1817 [67]	Trusted Player					✓
Decentralised Electronic Marketplace [85]	LGI and Controllers	✓		✓		
Idemix [13]	Anonymous Credentials	✓	✓			✓
Tagged Transactions	Digital Signatures and Verifiable Third Party	✓	✓	✓	✓	✓

Table 2.1: Comparison of Related Work

items with an offline supplier. The potato system takes in to account financial motivations and rewards for users of the system and can provide provenance information but uses a centralised trusted third party to verify reseller transactions.

The Decentralised and Secure Electronic Marketplace (DEM) takes the trusted third party and distributes it across a number of hosts. While the distribution of the agents provides better reliability with no central point of failure, there is still no method to verify the agents and they act as a black box. The owners of the controllers can also build up data on the transactions in the marketplace and what parties were involved. DEM can be used to provide provenance information for digital items.

Anonymous credentials provide mechanisms to anonymously establish provenance for digital items in reseller chains. Anonymous credentials provide both anonymity and unlinkability for customers and resellers. However, they do not provide anonymity for suppliers or support for offline suppliers.

2.1.11 Provenance in Web Services

Although there has been research in the area of provenance for web services [39, 89] there has been little work in providing provenance for web services in the presence of active adversaries. Hasan et al have constructed a protocol for securely collecting provenance information from distributed sources [43, 44]. Each document in the system has a provenance chain. This chain is composed of provenance records. When a document is modified, a new provenance record is added to the provenance chain for that document. A provenance record contains the identity of the principal modifying the document, a (possibly encrypted) representation of the actions performed on the document, a hash of the final value of the document, a hash of some of the contents of the provenance record and some of the contents of the previous record in the chain signed by the principal modifying the document (checksum), and a public key certificate of the principal signed by a trusted certificate authority.

The trust model in this scheme looks at both integrity and confidentiality properties. A set of auditors are used to check if provenance chains are plausible. The auditors are assumed to perform and report back the results of reports correctly. The integrity properties include detecting an adversary adding or removing provenance records from the provenance chain and detecting an adversary modifying the provenance chain. The confidentiality properties prevent

unauthorised auditors from accessing sensitive provenance records. Provenance records can be set to be accessible to a certain subset of auditors, but at least one auditor must be able to access every provenance record in the provenance chain.

Digital signatures and the hash values of the final document in the provenance records are used to detect tampering with the provenance records. If an adversary adds or removes provenance records from the middle of the chain, it will be detected as the signed checksums in the chain will not follow on from one another. To check that the provenance chain is for the correct document, an auditor checks the hash of the document in the final provenance record in the provenance chain is the same as the current version of the document.

The protocol for providing provenance in web services developed in this thesis and this protocol both use a similar data structure to represent provenance information. Both use a chain of provenance information composed of individual provenance records. In contrast to the work in this thesis, the threat model of the work by Hasan et al does not include exclusion attacks. Exclusion attacks involve an adversary taking some data and removing all provenance data from it and claiming it as their own. In this thesis I do not include the confidentiality requirements of the work by Hasan et al.

2.2 Summary

This chapter has presented the related work for provenance in reseller chains. An initial overview of systems that provide data provenance for geographic information systems, scientific workflows, databases, and tracking the source of attacks in Grid nodes. This overview shows that the area of digital provenance research is an expansive one. Provenance systems are an important part of many Grid computational systems. The Open Provenance Model (OPM) has been briefly described. The OPM has been applied to the problem of distributed computing in other work. In this thesis the OPM is used to model information for provenance of digital items in reseller chains and provenance in web services.

A survey of systems that can be used to provide provenance in reseller chains has been presented. No single system is able to fulfil all the requirements established in Section 1.2. Many of the protocols rely on some trusted third party. These range from a trusted computing module (TCM), to a trusted player, or a set of trusted controllers. The anonymous credential system can anonymously

establish provenance for digital items in reseller chains. However, anonymous credentials do not support anonymous suppliers or offline suppliers.

A protocol for providing provenance for web services has been presented. This protocol uses similar data structures and methods to the protocol developed in this thesis but does not discuss exclusion attacks.

Chapter 3

Domain and Threat Model

This chapter defines the domain, provenance, and threat models for anonymously establishing digital provenance in reseller chains. These models show in what domain the protocol operates, what provenance information is provided, and what threats the protocol protects against. Finally, I construct a formal definition of secure provenance.

The domain model shows the parties involved in the protocol. The roles of the parties in the protocol are then defined.

To provide digital provenance information in reseller chains, the required provenance information needs to be enumerated. The definitions and actions defined by the Open Provenance Model (OPM) are applied to model the problem of providing provenance in reseller chains.

Before designing the protocol, a threat model is needed to define any assumptions made about the participants in the protocol, and what environment the protocol is running in. For example, does the protocol assume passive or active adversaries? The threat model lists the assumptions about the attacks the protocol prevents and under what circumstances an attack is considered to be successful.

A security analysis of the resulting protocol requires a formal definition of secure provenance. This formal security definition is built from the definitions of the algorithms involved in the protocol and the possible attacks on the protocol.

3.1 Domain Model

A participant in the reseller domain takes one of three roles:

- The Supplier: Suppliers are the original creators or holders of the rights for an item.
- The Reseller: The reseller or middleman has a set of customers who purchase goods from them and a set of suppliers that the reseller purchases the goods from.
- The Customer: The customer purchases items from a reseller that are produced by a supplier.

A customer may change roles to become a reseller. For example, once a customer has purchased an item, it can on-sell this item, in which case the customer takes the role of the reseller. The changing of roles supports customer reselling of items.

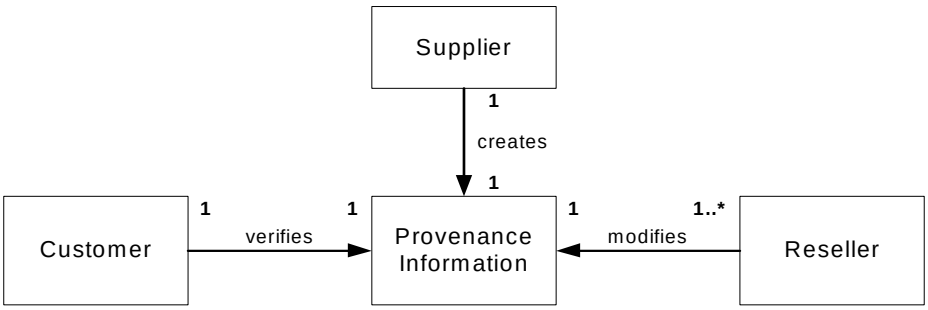

Figure 3.1: Domain Model

Figure 3.1 shows the domain model for provenance in reseller chains. A supplier creates provenance information for an item. For each transaction, a single supplier generates a single piece of provenance information. Resellers modify the provenance information generated by the supplier. Many resellers may modify a single piece of provenance information to support multiple resellers between the supplier and the customer. A single customer will verify a single piece of provenance information.

3.2 Provenance Model

The Open Provenance Model (OPM) defines a set of terms and relationships to record provenance of items [59]. These terms and relationships are used to create a provenance graph that shows the provenance information for an item. If a

verifier can reconstruct the provenance graph for an item, they can show the provenance information for the item. The following terms are defined:

- Artifact: An immutable piece of state.
- Process: An action or series of actions performed on or caused by an artifact.
- Agents: A contextual entity controlling a process.

The Open Provenance Model also defines actions used to model provenance information. The role of the process or agent performing the action parametrises the actions:

- wascontrolledby(Role): A process is controlled by an agent. The Role parameter is the role the agent is taking in the protocol.
- wasgeneratedby(Role): An artifact is generated by a process. The Role parameter is the process generating the artifact.
- usedby(Role): An artifact is used by a process. The Role parameter is the process that uses the artifact.

Using these terms and definitions, I construct a provenance model for the domain of reseller transactions. This model instantiates the OPM terms (as shown in Figure 3.2):

- Artifact: An artifact is an item that the supplier produces and sells, via resellers, to the customer.
- Process: A process is either "Create" where a supplier generates an item or "Resell" where a reseller resells an item.
- Agents: All agents are suppliers or resellers. Customers do not feature as agents as they do not create provenance information.

The constructed model of provenance information in reseller chains instantiates the OPM actions as:

- wascontrolledby(Role): The process of "Create" is controller by a supplier and "Resell" is controlled by a reseller. The roles are defined as either "Supplier" or "Reseller".

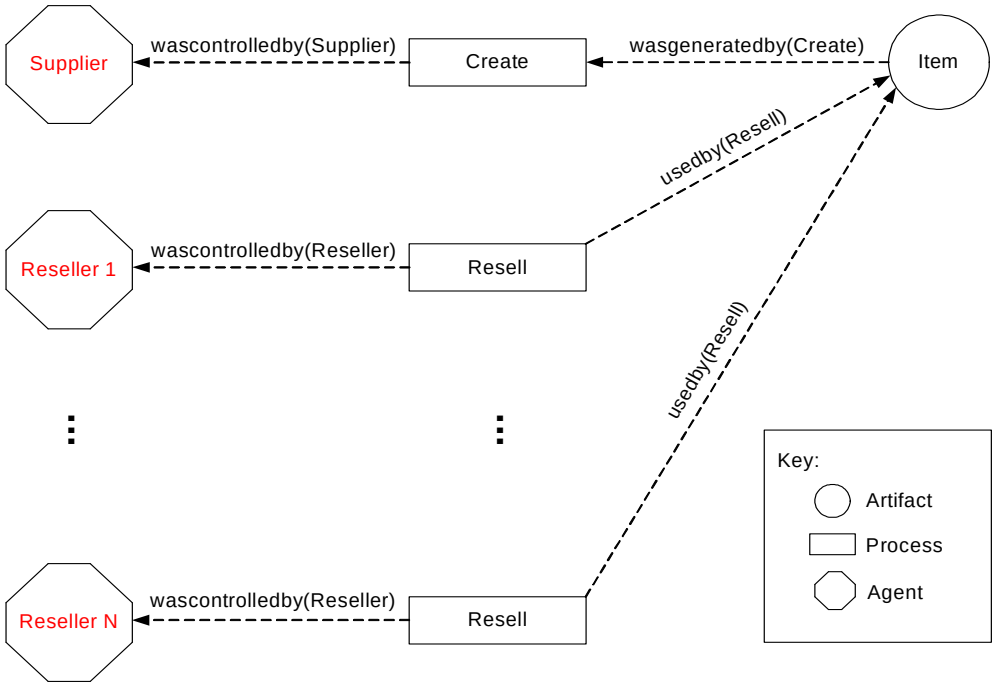

Figure 3.2: Provenance Graph for Tagged Transaction Protocol

- wasgeneratedby(Role): The item is generated by the supplier. The role is defined as "Create".

- usedby(Role): The item is used by the "Resell" process. The Role parameter is "Resell".

Figure 3.2 shows a provenance graph of provenance in reseller chains. The item is created by the create process that is controlled by the supplier. The item is then used by the resell process, controlled by a reseller, where the item is resold down the reseller chain.

To preserve the privacy of parties taking part in the transactions a verifier needs to be prevented from being able to reconstruct the entire provenance graph with all identities. However, to verify the provenance of the items purchased, a verifier needs to be able to have confidence that this provenance graph can be constructed. This work presents two methods of managing this conflict:

1. The customer reconstructs all the provenance information without learning the identities of the parties involved in the protocol, or any information to link together actions of a participant in different runs of the protocol.

2. The customer relies on a third party or a threshold value of a group of independent third parties to verify this provenance information and trusts the results it receives.

Chapter 6 describes these approaches as well as the architecture and verification of the TGC in more detail.

3.3 Threat Model

I make the following assumptions about adversaries and participants in the protocol design. The reseller is a polynomially bounded active adversary. The customer and supplier are also polynomially bounded. If the reseller is selling item x, then the reseller cannot collude with the supplier for item x, but can try to impersonate the supplier. If the reseller could collude with the supplier for item x, then the supplier could convince a customer that a malicious reseller is acting correctly. The verification of the reseller relies on the supplier wanting to prevent and, if possible, discover malicious resellers for the items it sells. While the reseller selling item x cannot collude with the supplier for x, the reseller may collude with any other supplier to try to convince the customer that the other supplier is the valid supplier for item x.

The customer does not collude with the reseller. If the customer was prepared to collude with the reseller, then they would have no incentive to take part in any verification protocol; either they would seek out a malicious reseller to avoid having to pay full price for the item, or they would acquire the item illegally.

While neither the supplier nor the customer collude with the reseller, it is not assumed that both are honest. Either the supplier or the customer may try to discover the identity of anonymous participants in the protocol. Additionally, either the customer or supplier may try and link together actions of other participants in the protocol. As the customer is able to resell the item after purchasing it from a reseller, when the customer plays the role of a reseller it should be prevented from being able to perform attacks on the protocol.

There may be a third party who tries to discredit a participant in the protocol by impersonating them. For example, a reseller r_1 may try to pose as another reseller r_2 and act in a malicious way so that the verification protocol fails and it appears r_2 is untrustworthy when it is really an act of sabotage by r_1.

The adversary follows a Dolev Yao model [30]. In the Dolev Yao model,

the adversary has full control of the network and can intercept messages, read messages, drop messages, or insert fake messages. Denial of service attacks, where an adversary may either drop all messages for a participant in the protocol or bombard a participant with too many messages, are not considered an attack unless they result in a spoofing, counterfeiting, identity revelation, or linkability attack.

Side channel attacks are not considered. There are many potential side channel attacks that could be made on a protocol for providing provenance in reseller chains. For example, if an adversary knows the most common times for a New Zealand participant to be involved in a protocol, the adversary may be able to gain some information about what country (or time zone) a participant is in based on the time of the protocol's execution. This is just one example and there are many varied side channel attacks that have been applied to many different security protocols.

As customers directly deal with resellers, a group of customers can link together the actions of a reseller with respect to this group. As the customers have to directly deal with the reseller to browse their store and arrange a purchase, there is no way to prevent the customers from learning the identities of the resellers they are purchasing items from. This is considered a side channel attack as the identity revelation is not a direct effect of the protocol to establish provenance.

When an anonymous communication channel is used, it is assumed to be a perfect anonymous channel. The design and implementation of anonymous communication channels is beyond the scope of this work.

3.4 Protocol Definition

In this section, I present a high level description of a protocol for anonymously establishing digital provenance in reseller chains. This high level description of the protocol is implementation independent; a variety of implementations could fulfil the requirements of the description. This work provides implementations of protocols that fulfil these requirements. The following notation describes the protocol. The value $item$ is a unique identifier for the item. The notation pk represents a public key and sk represents a private key. The variable $data$ is information that is used to generate the $license$ that verifies the provenance of

$item$.

A protocol for anonymously establishing digital provenance in reseller chains is defined as a set of five multi-party algorithms.

1. $Setup(k)$: a probabilistic polynomial time (PPT) algorithm that sets up keys and global parameters necessary for the protocol with security parameter k.

2. $Register(item, pk_{item})$: a polynomial time algorithm where the supplier for $item$ registers it with some public information pk_{item} where only the supplier knows the corresponding private information sk_{item}. Returns 1 or 0 to indicate success or failure of the registration.

3. $Generate(item, data, sk_{item})$: a PPT algorithm which returns $license$ for $item$. The algorithm generates $license$ using the secret information for the item sk_{item} and $data$.

4. $Regenerate(item, license_{old}, data_{old}, data_{new})$: a PPT algorithm which returns $license_{new}$ for $item$. The algorithm generates $license_{new}$ using $data_{new}$ as well as $license_{old}$ and $data_{old}$ from an existing license for $item$.

5. $VerifyLicense(item, data, license)$: a polynomial time algorithm that verifies the correctness of $license$ for $item$ and $data$ and returns 1 or 0.

The use of the $Generate$ and $Regenerate$ algorithms will result in a set of licenses $\{license_1, ..., license_n\}$ with corresponding data values $\{data_1, ..., data_n\}$. These values form a well defined sequence from $i = 1, ..., n$. The following formula express the correctness property:

$$VerifyLicense(item, data_i, license_i) = 1$$

$$\text{where } license_1 = Generate(item, data_1, sk_{item})$$
$$\text{and } license_i = Regenerate(item, license_{i-1}, data_{i-1}, data_i)$$

3.5 Security Properties

The security of a protocol for anonymously establishing digital provenance in reseller chains consists of ensuring security against spoofing, fabrication, cloning, network sniffing, identity revelation, and linkability attacks. Let k be a suitable security parameter as used in the $Setup$ method. As it is assumed the supplier

for $item$ is honest for transactions involving $item$, the $Setup(k)$ method generates $item$, pk_{item}, and sk_{item}.

The following definitions use the public (pk_{TGC}) and private key (sk_{TGC}) of the Tag Generation Centre (TGC) which is a third party used in the tagged transaction protocol. While these parameters are specific to the tagged transaction protocol they can be substituted for any other protocol specific values when using the following definitions for analysing other protocols. For the following definitions, the TGC acts as a trusted third party. Subsequent work removes this assumption (Chapter 6).

The notation $Prob[x]$ indicates the probability of action x. The notation $A^{a,b}(params)$ indicates the adversary has access to the oracles a and b and the data in $params$.

Oracles:

The adversary A has access to three oracles:

1. $O_{init}(item, pk)$ registers $item$ using pk. O_{init} simulates the $Register$ algorithm.

2. $O_{gen}(item, data)$ generates a license for $item$ using $data$. O_{gen} simulates the $Generate$ algorithm.

3. $O_{reg}(item, License_{old}, data)$ generates a new license for $item$ using $data$ and an old license $License_{old}$. O_{reg} simulates the $Regenerate$ algorithm.

3.5.1 Prevention of Spoofing

To claim to be the supplier, the adversary A will have to register the item. Any probabilistic polynomial time (PPT) adversary A should have negligible probability of completing the following experiment $exp_{spoofing}$:

$Setup(k) \rightarrow (sk_{TGC}, pk_{TGC}, item, pk_{item}, sk_{item})$
$pk_r \leftarrow A^{O_{init}}(pk_{TGC})$
$exp_{spoofing} = Prob[1 \leftarrow Register(item, pk_r)]$

where O_{init} cannot be used with $item$.

3.5.2 Prevention of Fabrication

An adversary A should not be able to produce a valid tag for $item$ and $data$ without knowledge of the secret key for the item sk_{item}. Any PPT adversary A should have negligible probability of completing the following experiment $exp_{fabrication}$:

$Setup(k) \rightarrow (sk_{TGC}, pk_{TGC}, item, pk_{item}, sk_{item})$
$Register(item, pk_{item}) \rightarrow 1$
$license \leftarrow A^{O_{gen}, O_{reg}}(item, data, pk_{item}, pk_{TGC})$
$exp_{fabrication} = Prob[1 \leftarrow VerifyLicense(item, data, license)]$

where O_{gen} and O_{reg} cannot be used with $item$ and $data$.

3.5.3 Prevention of Network Sniffing

In a network sniffing attack, the adversary sees a valid license on the network and uses it to generate a new license. Any PPT adversary A should have negligible probability of completing the following experiment $exp_{sniffing}$:

$Setup(k) \rightarrow (sk_{TGC}, pk_{TGC}, item, pk_{item}, sk_{item})$
$Register(item, pk_{item}) \rightarrow 1$
$lic_{old} \leftarrow Generate(item, data_{old}, sk_{item})$
$lic \leftarrow A^{O_{reg}}(item, data, pk_{item}, pk_{TGC}, lic_{old})$
$exp_{sniffing} = Prob[1 \leftarrow VerifyLicense(item, data, lic)]$

3.5.4 Prevention of Cloning

In a cloning attack, the adversary has access to a valid license which it uses to create two new valid licenses. Any PPT adversary A should have negligible probability of completing the following experiment $exp_{cloning}$:

$Setup(k) \rightarrow (sk_{TGC}, pk_{TGC}, item, pk_{item}, sk_{item})$
$Register(item, pk_{item}) \rightarrow 1$
$lic_{old} \leftarrow Generate(item, data_{old}, sk_{item})$
$lic_1 \leftarrow A^{O_{reg}}(item, data_1, pk_{item}, pk_{TGC}, lic_{old})$
$lic_2 \leftarrow A^{O_{reg}}(item, data_2, pk_{item}, pk_{TGC}, lic_{old})$

$exp_{cloning} = Prob[1 \leftarrow VerifyLicense(item, data_1, lic_1)]$ AND
$Prob[1 \leftarrow VerifyLicense(item, data_2, lic_2)]$

3.5.5 Prevention of Identity Revelation

In an identity revelation attack, the adversary A finds the identity of a participant in the protocol that is not its neighbour in the reseller chain. The adversary has access to the set M of all messages sent as part of the protocol and the set N of the identities of all parties involved in the protocol other than the TGC. The TGC is a well-known party that signs messages and is not involved in the transactions so it is excluded from the definition of identity revelation. The symbol id_m denotes the identity of the sender of the message m and $Prob[id_m]$ denotes the probability of identifying the sender of m. Identity revelation is defined in terms of A finding the identity of the sender of a message. Formally, any PPT adversary A should have negligible probability of completing the following experiment $exp_{identity}$:

$$\text{Given } m \in M$$
$$exp_{identity} = Prob[id_m] - \frac{1}{|N|}$$

3.5.6 Prevention of Linkability

Unlinkability ensures that a user may make multiple uses of resources or services without others being able to link these uses together (ISO 15408 standard [50]). In a linkability attack, the adversary A links together actions of a participant other than the supplier or the TGC based on the messages posted as part of the protocol. The supplier and the TGC are removed from the set of participants in this case. As the supplier first generates all licenses for an item, linking together the actions of the supplier is trivial. In fact, the protocol should ensure that the correct supplier generates the initial license for an item.

The symbol M denotes the set of all messages sent as part of the protocol and N denotes the set of the identities of all parties involved in the protocol other than the TGC and suppliers. The value $m_{i,r}$ is defined as the messages sent by participant i in run r of the protocol. Informally, the definition of unlinkability involves the adversary not being able to tell the difference between a set of messages sent by a random participant, and the set of messages sent by a participant that they have seen messages from before. The adversary is

given access to all messages M. Initially, the adversary is also given access to the identity of a participant i, and the messages $m_{i,r1}$ sent by participant i in run $r1$. A second participant is randomly chosen that is different to the first participant and a second run of the protocol that features both participants. Either 0 or 1 is chosen with equal probability. The adversary is then given the challenge of deciding whether a group of messages $m_{x_b,r2}$ was sent by participant i or j. Formally, any PPT adversary A should have negligible probability of completing the following experiment exp_{link}:

1. Select participant i, run $r1$, and $m_{i,r1}$
2. Choose $j \in_R N$ where $i \neq j$
3. Select run $r2$ which features participant i and j
4. Choose $b \in_R \{0, 1\}$
5. Set $x_b = i$ and $x_{1-b} = j$
6. $b' \leftarrow A(i, m_{i,r1}, m_{x_b,r2}, m_{x_{1-b},r2})$
7. $exp_{link} = Prob[b' = b] - \frac{1}{2}$

3.5.7 Security Definition

As a contribution of this work, I define the formal definition of security for providing digital provenance in reseller chains. The formal definition uses the definitions already presented to create a top level definition of secure provenance. This definition can then be used to analyse the protocols constructed in this Thesis to check they provide secure provenance.

Secure Provenance. *Any probabilistic polynomial time adversary (PPT) A should have negligible probability of completing a spoofing, fabrication, cloning, network sniffing, identity revelation, or linkability attack.*

3.6 Summary

In this chapter, a domain model for reseller chains was presented. This domain model shows the parties and roles involved in the protocol. A model and graph

of provenance information for the reseller model was presented in the Open Provenance Model format. This model shows the information necessary for a verifier to construct the provenance information for a digital item. The model also shows the conflict between providing provenance information and providing anonymity for participants. This conflict is resolved using one of two methods: removing the identifying information from the provenance information, and having a third party or group of independent third parties verify the provenance information.

An in depth threat model has been presented. The threat model explicitly states the assumptions made when constructing the protocol to establish provenance information in reseller chains. The threat model also shows what is considered an attack on the protocol and what parties may collude together.

I have defined a protocol for anonymously establishing digital provenance in reseller chains as a set of five algorithms. These algorithms describe any protocol that is designed to solve this problem and are not limited to the protocols created in this thesis. The security properties are then formally defined as prevention of spoofing, fabrication, cloning, network sniffing, identity revelation, and linkability attacks. These security properties are used in the formal definition of secure provenance. Using this definition, it can be shown whether a protocol to establish provenance information in reseller chains is secure.

The next two chapters present two protocols that provide secure provenance in reseller chains. The protocols are analysed against the specification provided in this chapter.

Chapter 4

Tagged Transaction Protocol

This chapter presents the design and analysis of the tagged transaction protocol for anonymously providing provenance of digital items in reseller chains. The tagged transaction protocol uses a third party called the Tag Generation Centre (TGC) to generate and sign tags. If a customer has a valid tag that is signed by the TGC then they can be confident in the provenance of the item that the reseller has sold them. The properties of the tagged transaction protocol include:

1. It provides a method for customers to check the provenance of an item they are purchasing, even from an untrusted reseller.

2. It provides anonymity and unlinkability for customers and resellers that are not neighbours in the reseller chain. It also provides optional anonymity for suppliers. This prevents other parties from building up information on the transactions of participants in the protocol.

3. It supports customer reselling of items. If a customer wishes to resell an item they have purchased from a reseller, they can take the role of a reseller and sell it to another customer.

4. It does not require the interaction of the supplier once the initial tag has been generated. The resellers and the TGC co-operate to generate the subsequent tags.

5. It does not provide enforcement of licenses.

The tagged transaction protocol does not include payment methods. The implementation of a fair payment scheme is out of the scope of this work. Chapter 5 describes an extended version of the tagged transaction protocol with

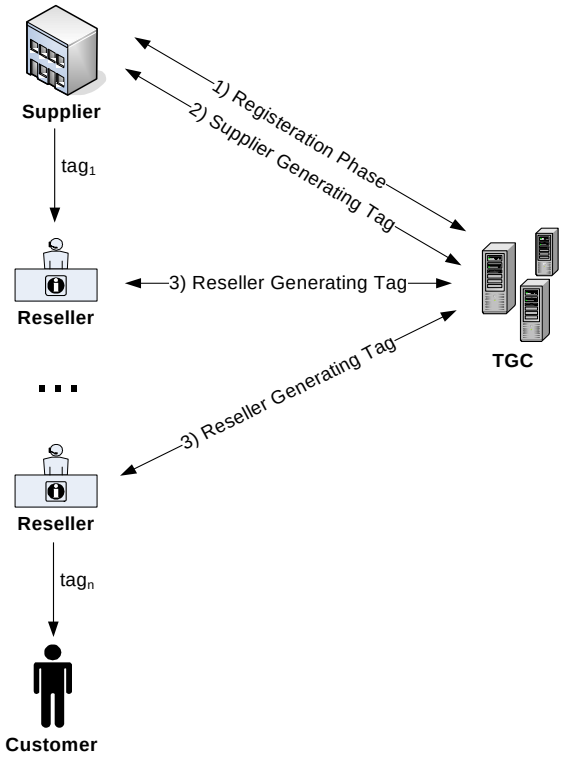

Figure 4.1: Overview of the Tagged Transaction Protocol

revocable anonymity. Chapter 6 describes mechanisms to verify the actions of the TGC while preserving the anonymity and unlinkability of the parties involved in the transaction.

The tagged transaction protocol provides secure provenance information at every step in a reseller transaction. When a customer or reseller receives provenance information they will want to authenticate the item (make sure that the item originally came from the correct supplier) and to authenticate the current owner (make sure the reseller or supplier they are purchasing the item from has a legitimate copy of the item with correct provenance information). While the customers and resellers will want to authenticate this provenance information the identities of the participants in the protocol should remain anonymous and unlinkable between different runs of the protocol.

Figure 4.1 shows the operation of the tagged transaction protocol. Suppliers and resellers create tags that are passed to the purchaser of an item, either a customer or another reseller. There can be many resellers in the chain between the supplier and the customer. The tagged transaction protocol has three main

phases as shown by the numbers in Figure 4.1:

1. Registration Phase. The supplier initially registers the item with the TGC.
2. Supplier Generating Tag. The supplier and TGC take part in a protocol to create a tag for an item.
3. Reseller Generating Tag. Resellers and the TGC take part in a protocol to update the information in the tag. The reseller presents the TGC with a valid tag before the TGC will update the information in that tag. This phase is repeated as many times as required depending on the number of resellers in the chain.

4.1 Definitions

In the tagged transaction protocol, tags are used to represent provenance information and are passed from the supplier to the customer via resellers. A tag is defined as a tuple:

$$tag = \{A = pk_x, B = L_x, C = pk_{tag,r}\}$$

The parameters of the tag are:

- $A = pk_x$: the public key for the item. This is registered with the TGC when the supplier first creates the item. The TGC should only sign tags that have the correct public key for the item in this entry.
- $B = L_x = \{id = H(x), License\}_{sk_x}$: a license signed with the secret key for the item x. This license contains information such as the identity of the item $id = H(x)$ and the other terms of the license. Section 4.1.1 discusses identifying items and Section 4.1.2 discusses the structure of licenses.
- $C = pk_{tag,r}$: the one time public key for the participant r and tag tag.

To preserve the anonymity of the supplier, resellers, and customers, all communication with the TGC must be done over an anonymous communication channel. Section 6.1 discusses mechanisms to implement the anonymous communication channel. The signed license links this tag with the item being sold. The license contains information to allow any party to check what item the license is for and the terms of the license. A one time key is used for tags as a

single key for each participant links together the transactions of that participant in multiple runs of the protocol. Resellers and customers generate one time public and private keys and the public key is embedded in the tag. For tag tag and reseller r, the value $pk_{tag,r}$ denotes the public key and $sk_{tag,r}$ denotes the secret key.

The tagged transaction protocol uses a digital signature scheme and an encryption scheme. The TGC signs tags and the public signing key of the TGC has to be well known to verify the signed tags. The public encryption key of the TGC is also well known so that the supplier can encrypt messages to send to the TGC. Before the protocol is run for the first time, the TGC generates and publishes public encryption and signing keys. The notation $\{A\}_{sk_B}$ denotes the message A signed using the key sk_B and $\{A\}_{pk_B}$ denotes the message A encrypted using the key pk_B.

4.1.1 Identifying Items

To establish provenance of digital items, items need to be uniquely identified. The requirement of identifying digital items is complicated by the fact that digital items could be modified, either maliciously or as a requirement of functionality. For example, a music file may be available for sale from a reseller in one of several formats. To transfer between these formats, a new encoding is applied to the file and the music file needs to be recognisable as the same digital item under these transformations. Other transformations may be applied maliciously. For example, a malicious reseller could cut a small period from the start or end of a music file, or alternately they could add a small period to the start or end of the song to try and make it appear as a new item that they can claim ownership of. There are several techniques used to identify items ranging from simple techniques such as hash values and string values, to stenographic techniques such as digital watermarking.

Hash Values

A hash value is a basic form of identification where a cryptographic hash function H, such as MD5 [76] or SHA-1 [32], is applied to the item x to produce the identity $id = H(x)$. A hash value is not robust under modifications or transformations of the item it identifies. Any change to the item will result in a new identity. Hash

functions are efficient to compute, a customer can easily check if the item they have received is identified by a specific hash value.

String Values

A digital item may also be identified using string values. In its most basic form a string value can be the title of the digital item, such as the name of a song. A more complex representation of an item using string values may include extra meta data such as the artist who recorded the song, the year of release, and the album name. Meta data values are dependent on the type of item being identified, with different types of items having different meta data. String values in the form of XML files are used in the ISO standard 21000-3 Digital Item Identification [51]. An XML file is created that provides a description of the item and an identifier in a standard identification scheme such as ISBN (International Standard Book Number) or ISRC (International Standard Recording Code).

String values are robust under transformations applied to the file as they are not calculated from the data of the item. When using string values to identify items, customers may not know the exact name identifying an item. For example, when given two different identifiers "Windows XP" and "Microsoft Windows XP" it may be difficult for the customer to choose the correct identifier of the item they wish to purchase.

Digital Watermarking

Digital watermarking is a stenographic technique that embeds information in data by applying minor modifications to the data in a perceptually invisible manner. The watermark information can be recovered from the modified data by detecting the presence of these modifications. There have been many digital watermarking schemes that have been put forward in the literature [9, 10, 24, 26, 42, 71, 84, 90]. There are many different techniques used to watermark media files including least significant bit substitution [9, 26, 90], transform domain techniques [10, 24], and spread spectrum techniques [42].

Digital watermarking may be symmetric or asymmetric. In a symmetric watermarking system, the same key is used to both embed and detect a watermark. In asymmetric watermarking, a different key is used to embed the watermark than is used to detect it.

Digital watermarking requires the modification of the item to be watermarked.

Some items, such as a piece of software, may no longer function correctly when they have been modified with a watermark.

A robust watermarking scheme preserves the watermark when different types of transformation are applied to the watermarked item. Digital watermarks have been shown to be secure against certain attacks to remove the watermark. However, digital watermarks are vulnerable to re-watermarking where an adversary applies a new watermark to the data without removing the existing watermark.

Discussion

Uniquely identifying items is a difficult problem, especially when there are a large number of different types of items that may have separate transformations applied to them. There has been a lot of research work in the area [9, 10, 24, 26, 42, 51, 71, 84, 90]. In general the problem of uniquely identifying items is beyond the scope of this thesis. However, in the protocols constructed in this thesis a hash value is used as the identifier for an item. While hash values have problems identifying items that have had a transformation applied to them, it is a simple and efficient solution. When a single item has multiple different representations, a group of hash values can be used to show an item is one of a group. Any identification scheme could be substituted for the hash value, the constructed protocols do not require a specific identification scheme.

4.1.2 Structure of Licenses

In the tagged transaction protocol, the supplier creates and signs a license for the digital item being sold. This license should contain information to identify the item, as well as any extra terms of the license such as duration of the license. The main restriction on the license information is that the license cannot contain any identifying information of the supplier, reseller, or any other parties in the protocol. The exception to this rule is if the supplier is not anonymous, in which case the license can include the identity of the supplier. The license may be a well known license, a pre-defined license, or a one off license. A well known license is a license that is used for many different items and is well known such as the GNU General Public License (GPL) [1]. A pre-defined license is a license that is applied to many items of the same sort such as the End User License Agreement

[1] http://www.gnu.org/copyleft/gpl.html

for Windows XP Home [2]. A one-off license is created fresh each time for one user and one item.

As the tagged transaction protocol sends license information separately from the digital items themselves, there is no need to embed the license information in the items. The license information can be in a separate data structure. A language for expressing license terms is a Rights Expression Language (REL). A REL presents rights information in an unambiguous machine readable format. The basic construct of a REL is a rights expression. A rights expression describes some permission granted for some protected content and may include conditions on the use of this permission. Several RELs have been developed including the creative commons Rights Expression Language (ccREL) [2], Open Digital Rights Language (ODRL) [65], and the MPEG-21 REL [52].

ccREL uses triples in the Resource Description Framework (RDF) format. RDF is a framework for describing entities on the web and uses URLs (or URIs) to address entities. An RDF triple for describing the license on a web page would have the URL of the web page, a URL defining the term license, and a URL pointing to the license for the web page. RDF triples can be represented in various ways but a common method is the use of XML. ccREL also provides definitions of features for providing information on the licensed content as well as any requirements or prohibitions on the license. While the ccREL is primarily designed to work with the creative commons license, it can be used with any type of license.

ODRL uses a Policy object as its core structure [65]. This policy can have attributes that describe the license such as permissions and prohibitions. A policy can be structured as an offer where rights are granted if the party performs certain duties. These duties can be actions such as paying a set amount for the permission to play a media file. Once the policy has been accepted, an agreement policy is created. This agreement policy must contain an identifier for the party the agreement is with. This party field would invalidate our requirement for anonymity of the parties in the protocol. However, a policy can also be created in a ticket form which grants the rights in the policy to the holder of the ticket. ODRL policies can be expressed in XML.

The core element of the MPEG-21 REL is a license [52]. A license contains one or more grants and an issuer. The issuer is the party that issued the license and signs the license and timestamps the license. One or more issuers may sign

[2] http://www.microsoft.com/windowsxp/eula/home.mspx

the same license. A grant comprises four parts: a principal the grant is issued to, the resource the grant is for, the right the principal has on the resource, and any conditions on the grant. To provide anonymity for the tagged transaction protocol, the principal would need to be generic or obfuscated in some way. The MPEG-21 REL can be expressed in XML.

Any of the RELs examined here can fulfil the requirements to express licenses in the tagged transactions protocol. Licenses can be formatted in XML documents that are sent to the customers. The choice of which REL to use can be left to the supplier. There are no restrictions on the type of license, other than the lack of identifying information. The supplier will want to choose a REL that is well-supported so that it can be sure that customers will be able to interpret the license when they receive it. The choice of REL may also be influenced by the type of license, for example, a supplier that uses a creative commons license may want to use ccREL as its design is closely aligned to the creative commons license.

4.1.3 Digital Signature Scheme

The tagged transaction protocol requires a digital signature scheme that is secure against existential forgeries under adaptive chosen message attacks. There are several well known digital signature schemes that provide this security property. The main signature schemes fall in to two families: El-Gamal based signatures, and RSA based signatures.

Tahir El-Gamal introduced El-Gamal signatures in 1984 [37]. The El-Gamal family of signatures rely on the difficulty of computing discrete logarithms over finite fields. The original signature scheme is subject to existential forgery attacks. Pointcheval et al developed a modified version of the signature scheme that is provably secure in the random oracle model against existential forgeries under adaptive chosen message attacks [74]. Schnorr developed a variant of the El-Gamal signature scheme that produces a shorter signature [83]. The Schnorr signature scheme has also been shown to be secure in the random oracle model against existential forgeries under an adaptive chosen message attacks [74]. Another popular variant of the El-Gamal signature scheme is the Digital Signature Standard (DSS) defined in the document FIPS 186-3 [63]. The DSS requires more computation to compute and verify signatures than either the modified El-Gamal or the Schnorr signature schemes.

The RSA family of signatures was introduced by Rivest et al in 1978 [77]. It

was the first practical implementation of a digital signature scheme. The family of RSA signatures relies on the difficulty of factorising large numbers. The original RSA signature scheme was subject to selective chosen message attacks. Further work incorporated a randomised padding scheme known as the probabilistic signature scheme (PSS) [7]. The PSS scheme makes use of two hash functions and the RSA function to produce randomised signatures. The PSS scheme has been shown to be secure in the random oracle model against existential forgeries under adaptive chosen message attacks and is the basis for the PKCS signature scheme [80]. The RSA-PSS signature scheme is more efficient than the El-Gamal family of signatures, requiring only one modular exponentiation and the execution of hash functions to sign and verify messages. The size of the keys in the RSA-PSS signature scheme is twice the size of the keys in the El-Gamal family of signatures. The PKCS RSASSA-PSS digital signature scheme is a well known implementation of the RSA-PSS signature scheme produced by RSA Security [80].

The tagged transaction protocol uses the Schnorr digital signature scheme [83]. This scheme has been shown to be secure against existential forgeries under an adaptive chosen message attack. The Schnorr signature scheme has the shortest signature size of the El-Gamal family of signatures and requires one less modular exponentiation to verify a signature than the modified El-Gamal scheme. The RSA-PSS signature scheme has shorter signature lengths and requires only one modular exponentiation to sign and verify signatures. However, once the initial system wide parameters for the Schnorr signature scheme are set, generating a new key pair only requires one modular division. The RSA-PSS scheme requires two large primes to be generated to create a key pair. As the tagged transaction protocol makes use of one time keys, the complexity of the key generation algorithm is an important factor.

4.1.4 Encryption Scheme

The tagged transaction protocol requires an encryption scheme that provides indistinguishability under adaptive Chosen Ciphertext Attacks (IND-CCA2). The two most commonly used encryption schemes are the RSA encryption scheme and the El-Gamal encryption scheme.

The RSA encryption scheme was developed by Rivest et al in 1978 [77]. The RSA cryptosystem relies on the difficulty of factorising large numbers. The

original RSA scheme was subject to chosen plaintext attacks. To provide IND-CCA2 security, various padding schemes have been developed. The most common of these are Optimal Asymmetric Encryption Padding (OAEP) [6] and the Probabilistic Signature Scheme with message Recovery (PSS-R) [8, 23]. RSA with the OAEP padding scheme, RSA-OAEP, has been shown to be IND-CCA2 secure in the random oracle model [6, 36, 86]. PKCS RSAES-OAEP is a well known encryption standard based on RSA-OAEP produced by RSA Security [80].

The first efficient public key cryptosystem with IND-CCA2 security was the Cramer Shoup public key cryptosystem [25]. The Cramer Shoup cryptosystem is an IND-CCA2 enhancement of the El-Gamal cryptosystem [37]. To provide IND-CCA2 security, the Cramer Shoup cryptosystem uses a hash value and adds several extra modular exponentiations to both encryptions and decryptions. General methods for converting one way trapdoor functions to IND-CCA2 encryption schemes have also been developed [35, 73]. These general methods can be applied to encryption schemes, such as the El-Gamal encryption scheme, to provide IND-CCA2 security and are more efficient than the Cramer Shoup cryptosystem.

In the tagged transaction protocol, I make use of the PKCS RSAES-OAEP encryption scheme. PKCS RSAES-OAEP has been shown to be IND-CCA2 secure in the random oracle model. To encrypt and decrypt in the PKCS RSAES-OAEP encryption scheme requires just one modular exponentiation. This is more efficient than any of the El-Gamal based encryption schemes. While the key generation for PKCS RSAES-OAEP requires more computation than for El-Gamal based schemes, the key generation only needs to be done once by the TGC. The tagged transaction protocol performs many more encryption and decryption operations than key generation making the efficiency of the encryption and decryption operations more important than the efficiency of the key generation.

4.2 Registration Phase

This section presents the registration phase of the tagged transaction protocol. There are two separate registration options. The first option provides anonymity for suppliers while the second option does not support anonymous suppliers. The registration process is shown in Figure 4.2. Both of the options involve the supplier first producing the item x. The supplier will then generate the identity

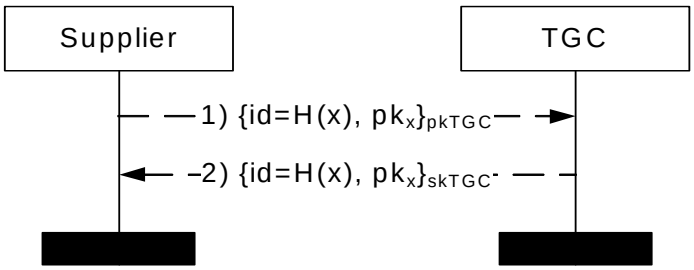

Figure 4.2: Registering the Item with the TGC

of the item $id = H(x)$ and a secret key (sk_x) and public key (pk_x) for the item. The public key and identity of the item are then registered with the TGC by sending the TGC the message $\{id, pk_x\}_{pk_{TGC}}$. The message is encrypted with the public encryption key of the TGC. This is to prevent an adversary from intercepting the message, changing the public key for the item and sending the message on to the TGC. After a successful registration the TGC returns a signed receipt $\{id, pk_x\}_{sk_{TGC}}$ or if the registration fails the TGC returns 0. If an anonymous supplier is required, the communication between the supplier and TGC must be done over an anonymous communication channel.

The anonymous supplier option uses a first-in first-registered style of registration. When the supplier is anonymous there are no checks the TGC can do to verify that the supplier registering the item is the correct supplier. When supplier anonymity is required the supplier should register the item before making it public to prevent an adversary registering the item before the supplier.

When supplier anonymity is not required the TGC may convince itself with out of band checks that the party registering the item is the correct supplier similar to checks done by a certification authority. This approach prevents denial of service, or brute force, attacks on the registration step where an adversary floods the TGC with registration requests in the hope of registering a real item.

4.3 Supplier Generating Tag with TGC

To initially generate a tag a protocol takes place between the reseller, the supplier, and the TGC. The communication between the supplier and the TGC is done over an anonymous channel if supplier anonymity is required as shown by dotted lines in Figure 4.3. The generation of a new tag for an item by the supplier takes

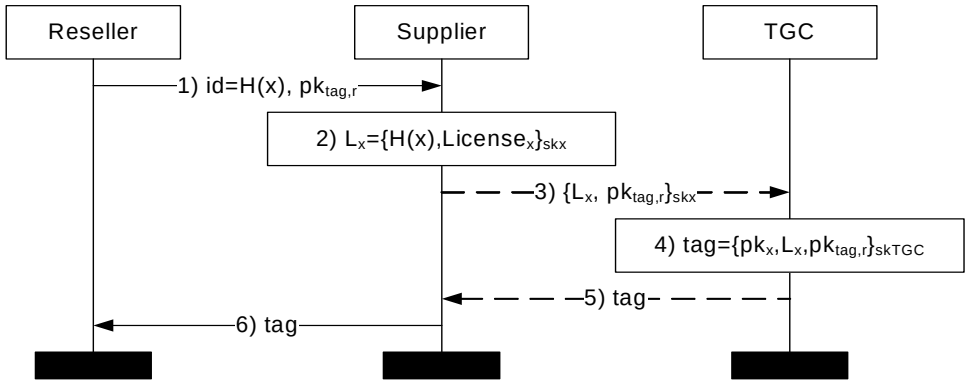

Figure 4.3: Supplier Generating Tag with TGC

place in the six steps shown in Figure 4.3.

1. The reseller sends a purchase request to the supplier containing the identity of the item they wish to purchase $id = H(x)$ and the one time public key for the tag $pk_{tag,r}$. The reseller randomly picks a private key and uses this to generate the public key.

2. The supplier signs a license for the item $L_x = \{id = H(x), License_x\}_{sk_x}$ where $id = H(x)$ is the identity of the item and $License_x$ is the license for the item.

3. The supplier then creates a signed tag request containing the license L_x and the one time public key $pk_{tag,r}$ all signed by sk_x and sends it to the TGC.

4. The TGC checks the one time public key $pk_{tag,r}$ has not been used for this item before and that the key is correct for this item. If these tests pass, the TGC constructs and signs $tag = \{pk_x, L_x, pk_{tag,r}\}_{sk_{TGC}}$.

5. The TGC sends the tag to the supplier.

6. The supplier sends the tag to the reseller. The reseller checks the tag has been signed by the TGC, that the license is for the correct item, and that the tag contains the correct one time public key.

The supplier signs the license generation request with the secret key for the item to prevent any other party from being able to generate tags for the item. The reseller will only accept the tag if the tag is signed by the TGC and contains the correct one time key.

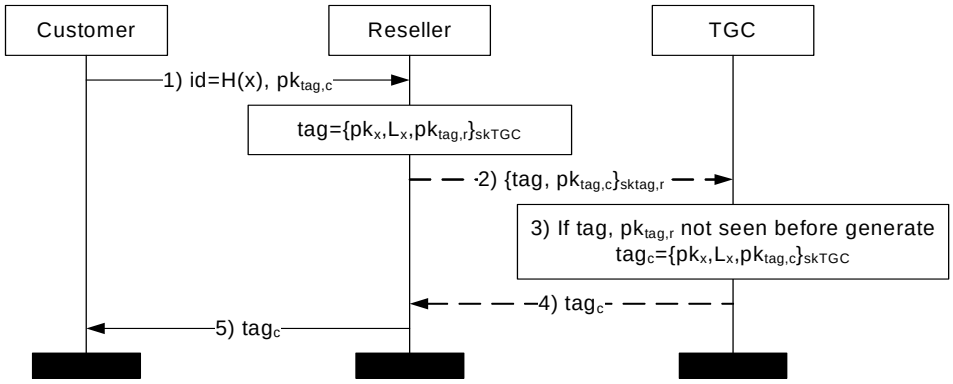

Figure 4.4: Reseller Generating Tag with TGC

4.4 Reseller Generating Tag with TGC

Now the tag for an item has been generated, signed, and sent to the reseller, the reseller can use this tag to generate a new tag for a customer or another reseller without interacting with the supplier.

The generation of a new tag by a reseller takes place in the five steps shown in Figure 4.4. The communication between the reseller and the TGC is done over an anonymous channel shown by dotted lines.

1. The customer sends a purchase request comprised of the identity of the item they wish to purchase, and the one time public key for the tag $pk_{tag,c}$. The customer randomly picks a private key and uses this to generate the public key.

2. The reseller sends a message with the one time public key chosen by the customer $pk_{tag,c}$ and the tag signed using the one time secret key used in the generation of the tag $sk_{tag,r}$ to the TGC.

3. The TGC then checks whether the tag has been cloned. The tag signed by the TGC contains a one time public key from the reseller. When the TGC generates a new tag for a customer it records the signed message from the reseller $\{tag, pk_{tag,c}\}_{sk_{tag,r}}$. If a reseller tries to clone a tag then the TGC will have two signed messages $\{tag, pk_{tag,c1}\}_{sk_{tag,r}}$ and $\{tag, pk_{tag,c2}\}_{sk_{tag,r}}$. The TGC presents these two signed messages as evidence the tag has been cloned. If the tag has not been cloned the TGC generates a new tag $tag_c = \{pk_x, L, pk_{tag,c}\}_{sk_{TGC}}$.

4. The TGC sends tag_c to the reseller.

5. The reseller sends tag_c to the customer.

The message $\{tag, pk_{tag,c}\}_{sk_{tag,r}}$ from the reseller to the TGC is signed using the one time private key for the tag. A reseller will need to know the one time private key for the tag to generate a new tag with the TGC. This prevents a network sniffing attack where an adversary sees a tag being sent over the network and tries to use it to generate a new tag. The signing of the message also prevents another reseller from being able to frame the original reseller in a cloning attack. As the message needs to be signed by the one time private key another reseller cannot modify the message sent to the TGC to alter the customer's one time key, or to fabricate a new tag signed using the one time private key. The signing of the message also allows the TGC to prove to a third party that a tag has been replayed by presenting the two messages $\{tag, pk_{tag,c1}\}_{sk_{tag,r}}$ and $\{tag, pk_{tag,c2}\}_{sk_{tag,r}}$.

This step of the tagged transaction protocol is repeated as many times as required. The customer will take the role of the reseller and generate a new tag for another reseller or customer further down the chain of resellers.

4.5 Security Analysis

Thereom 1. *The tagged transaction protocol provides Secure Provenance (Section 3.5.7) in the random oracle model provided that the signature scheme used has provable security against existential forgeries under adaptive chosen message attacks and the encryption scheme used has provable security against IND-CCA2 attacks.*

I use a reduction to contradiction style of argument to show the tagged transaction protocol provides security against spoofing, fabrication, network sniffing, and cloning attacks. I then make arguments showing the security of the tagged transaction protocol against identity revelation and linkability attacks. In this security analysis, the TGC is assumed to be acting as a trusted third party. Chapter 6 removes this assumption and discusses methods to verify the actions of the TGC. For the security analysis of the identity revelation and linkability properties the following assumptions are made: a perfect anonymous communication channel, an anonymous supplier, and the parties in the protocol not revealing their identities or the identities of the parties with whom they communicate. This security analysis does not consider side channel attacks.

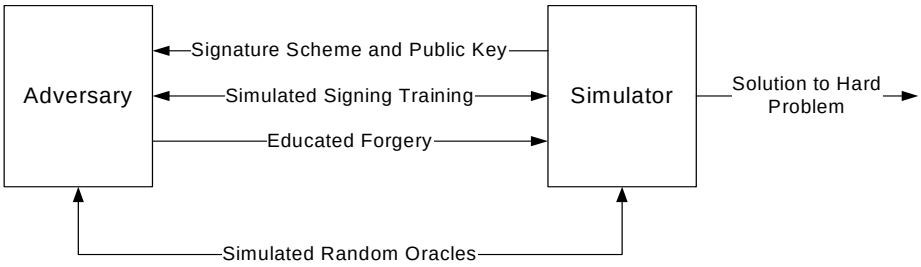

Figure 4.5: Simulator and Adversary for Signature Schemes

If the signature scheme provides security against existential forgeries under adaptive chosen message attacks in the random oracle model, then a simulator can be constructed that uses an adversary that breaks the signature scheme to solve a hard problem. This argument for the Schnorr signature scheme was first made by Pointcheval et al [74]. Figure 4.5 shows the setup for the random oracle model reduction. There are two parties, a simulator and an adversary. The adversary is assumed to be able to have a non-negligible advantage of breaking the signature scheme by outputting a valid message signature pair. The simulator provides the signature scheme and public key to the adversary. The simulator provides a signing oracle to the adversary by signing messages sent by the adversary. The input that the simulator provides to the adversary is computationally indistinguishable from the input it would receive from messages signed by a party with knowledge of the private key. The argument is then by contradiction. If the adversary can break the signature scheme, then the simulator can use the output of the adversary to solve a hard problem (in this case the discrete logarithm problem). Therefore, such an adversary cannot exist if the problem is hard. The simulator has no knowledge of the inner workings of the adversary which it treats as a black box. It is also important to note that the simulator does not have access to the private signing key for the signature scheme, however, it is able to simulate the signing in a computationally indistinguishable way from a real signer. All input to and output from the adversary goes through the simulator.

Figure 4.6 shows the setup for the tagged transactions reduction argument in the random oracle model. The simulator is the same as in the analysis of the security scheme. The adversary in this case is replaced by a tagged transaction simulator and a tagged transaction adversary. The simulator provides the same input to the tagged transaction simulator, namely a signature scheme and a

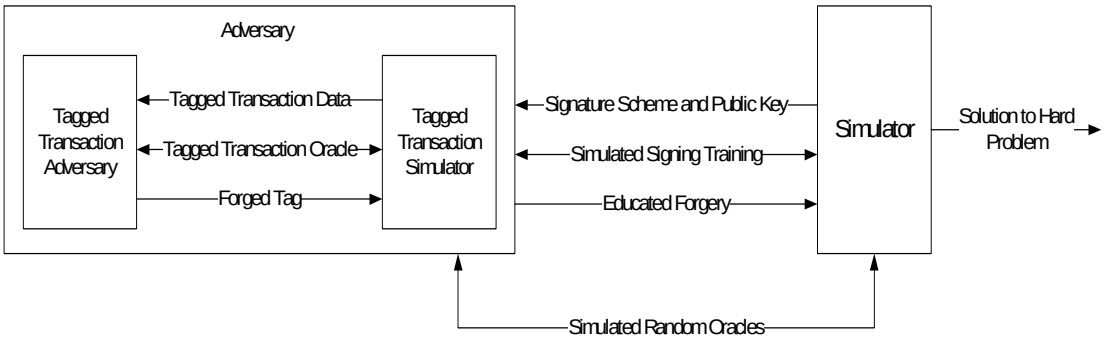

Figure 4.6: Simulator and Adversary for Tagged Transactions

public key. The simulator also provides simulated signature queries to the tagged transaction simulator. The tagged transaction simulator provides input to the tagged transaction adversary that is computationally indistinguishable from the input it would receive in an actual run of the tagged transaction protocol. If the adversary can break the security of the tagged transaction protocol, then the tagged transaction simulator can output a valid message signature pair for the signature scheme. If the tagged transaction simulator can output a valid message signature pair, then the original simulator can solve a hard problem. While the argument I have made corresponds to the signature scheme, a similar argument can be made regarding the encryption scheme.

The tagged transaction simulator provides the tagged transaction adversary with inputs that are computationally indistinguishable from the inputs it would receive in an actual run of the protocol. If the tagged transaction adversary breaks the security of the tagged transaction protocol in this simulated run, then the tagged transaction simulator can provide an output that can be used to solve one of two problems thought to be hard:

1. The tagged transaction simulator breaks the security of the signature scheme. When given public key pk and access to a signing oracle provided by the original simulator, the tagged transaction simulator outputs a valid message signature pair (m, σ) with non negligible probability where m was not used in a query to the signing oracle.

2. The tagged transaction simulator breaks the security of a IND-CCA2 secure cryptosystem. The tagged transaction simulator is given a public key pk and access to the encryption and decryption oracles provided by the original

simulator. The tagged transaction simulator can then make multiple calls to the encryption and decryption oracles. The tagged transaction simulator then generates two values m_0 and m_1 that have not been used in queries to the oracles and sends them to the original simulator. The original simulator generates $b \in_R \{0,1\}$ and $c_b = E(m_b)$ and passes c_b to the tagged transaction simulator. The tagged transaction simulator can break the IND-CCA2 cryptosystem if it can guess the value b with probability greater than $\frac{1}{2}$.

When providing inputs to the tagged transaction adversary and analysing the identity revelation and unlinkability properties, the security arguments involve showing computational indistinguishability of elements in a group. For a cyclic group $\mathbb{G}$ of order q with generator g, when given an element z chosen at random from $\mathbb{Z}_q$, the element g^z is uniformly distributed in $\mathbb{G}$. Two elements uniformly distributed in $\mathbb{G}$ are computationally indistinguishable. If two elements a_1 and a_2 are chosen at random from $\mathbb{Z}_q$, the elements g^{a_1} and g^{a_2} are computationally indistinguishable. If two private keys for the Schnorr signature scheme are chosen at random, the public keys are computationally indistinguishable. A random element from $\mathbb{G}$ and a Schnorr public key are computationally indistinguishable.

4.5.1 Spoofing

In a spoofing attack, the tagged transaction simulator is given as input the public key for the encryption scheme pk_e and access to the encryption and decryption oracles provided by the original simulator. The tagged transaction simulator is then challenged to break the security of the encryption scheme. The tagged transaction simulator generates the values id_0, id_1, and pk_{item} randomly and constructs $m_0 = (id_0, pk_{item})$ and $m_1 = (id_1, pk_{item})$. The tagged transaction simulator then sends m_0 and m_1 to the original simulator which returns $c_b = \{m_b\}_{pk_e}$ where $b \in_R \{0,1\}$.

Oracles:

The tagged transaction simulator implements the O_{init} oracle. The tagged transaction simulator records in a list the input values (id_i and pk_i) it receives. When a query is submitted that uses inputs already in the list, the tagged transaction simulator returns 0. If a query is submitted that does not appear in the list the tagged transaction simulator sends the message $\{id_i, pk_i\}$ to the encryption oracle which will return $\{id_i, pk_i\}_{sk_e}$.

Input:

The tagged transaction simulator provides input to the adversary that is computationally indistinguishable from the input it would receive in an actual run of the protocol. The tagged transaction simulator provides as input to the adversary $pk_{TGC} = pk_e$. The tagged transaction simulator also provides as input to the adversary $c_b = \{id_b, pk_{item}\}_{pk_e}$. As the adversary is assumed to be in the Dolev Yao model, they have complete control over the network and can intercept the registration message sent from the supplier to the TGC. The message $c_b = \{id_b, pk_{item}\}_{pk_e}$ is computationally indistinguishable from the real message sent from the supplier to the TGC.

Adversary:

Suppose adversary A passes the experiment with non negligible probability. The probability of the adversary guessing $item$ is negligible (2^{-k} where the output space of the hash function is k). As the adversary succeeds with non negligible probability, it must have modified the message $c_b = \{id_b, pk_{item}\}_{pk_e}$ to $c_A = \{id_b, pk_A\}_{pk_e}$ to register the item. The tagged transaction simulator queries the decryption oracle with the message $c_A = \{id_b, pk_A\}_{pk_e}$. If c_A decrypts to (id_0, pk_A) then $b = 0$ otherwise $b = 1$.

4.5.2 Fabrication

In a fabrication attack, the tagged transaction simulator is given as input the public key for the signature scheme pk and access to the signing oracle provided by the original simulator. The tagged transaction simulator is then challenged to break the security of the signature scheme by outputting a valid message signature pair. The tagged transactions simulator generates the values $item$, $data = pk_r$, and sk_{item} randomly using the random oracle of the original simulator. As $item$ is the output of a hash function it is computationally indistinguishable from a random value. The value sk_{item} is then used to generate the value pk_{item}.

Oracles:

The tagged transaction simulator implements the O_{gen} and O_{reg} oracles. To simulate O_{gen} in a manner that is computationally indistinguishable from the real protocol, the tagged transaction simulator records in a list the input values it receives. The adversary will send the message $\{id, pk_{tag,r}\}$ where $item_i = id$ and $data_i = pk_{tag,r}$. If the values $item_i$ and $data_i$ have not been used before the tagged transaction simulator submits the message:

$$m = \{pk_{item}, L_x, data_i = pk_{tag,r}\}$$

to the signing oracle and returns:

$$tag = \{pk_{item}, L_x, data_i = pk_{tag,r}\}_{sk}$$

The value $L_x = \{item, License_x\}_{sk_{item}}$ can be generated by the tagged transaction simulator as it has access to the values $item$ and sk_{item}.

To simulate O_{reg} in a manner that is computationally indistinguishable from the real protocol, the tagged transaction simulator records in a list the input values it receives. The adversary will send the message:

$$\{\{pk_{item}, L_x, pk_{tag,r}\}_{sk_{TGC}}, pk_{tag,c}\}_{sk_{tag,r}}$$

where $item_j = item$ in L_x, $data_j = pk_{tag,c}$, and

$$lic_{old,j} = \{pk_{item}, L_x, data_{old} = pk_{tag,r}\}_{sk_{TGC}}$$

If the values $item_j$, $lic_{old,j}$, and $data_j$ have not been used before and the message is correctly signed with $sk_{tag,r}$, the tagged transaction simulator submits the message:

$$\{pk_{item}, L_x, data_j = pk_{tag,c}\}$$

to the signing oracle and returns:

$$\{pk_{item}, L_x, data_j = pk_{tag,c}\}_{sk_{TGC}}$$

The value $L_x = \{item, License_x\}_{sk_{item}}$ is the same value that appears in $lic_{old,j}$.

Input:

The tagged transaction simulator provides input to the adversary that is computationally indistinguishable from the input it would receive in an actual run of the protocol. The adversary is given as input the values $item$, $data$, pk_{item}, and $pk_{TGC} = pk$.

Adversary:

Suppose adversary A passes the experiment with non negligible probability. The adversary is then able to fabricate a valid tag with non negligible probability:

$$tag = \{pk_{item}, L_x, pk_r\}_{sk}$$

The simulator can then return this to the original simulator as it has a valid message signature pair:

$$(\{pk_{item}, L_x, pk_r\}, tag)$$

The value $\{pk_{item}, L_x, pk_r\}$ will not have been used in a query to the signing oracle as neither the O_{gen} or O_{reg} oracles can be used with the value $data = pk_r$.

4.5.3 Network Sniffing

In a network sniffing attack, the tagged transaction simulator is given as input the public key for the signature scheme pk and access to the signing oracle provided by the original simulator. The tagged transaction simulator is then challenged to break the security of the signature scheme by outputting a valid message signature pair. The tagged transactions simulator generates the values $item$, sk_{TGC} and sk_{item} randomly using the random oracle of the original simulator. As $item$ is the output of a hash function it is computationally indistinguishable from a random value. The values sk_{item} and sk_{TGC} are then used to generate the values pk_{item} and pk_{TGC}.

Oracles:

The tagged transaction simulator implements the O_{reg} oracle. To simulate O_{reg} in a manner that is computationally indistinguishable from the real protocol, the tagged transaction simulator records in a list the input values it receives. The adversary will send the message:

$$\{tag = \{pk_{item}, L_x, pk_{tag,r}\}_{sk_{TGC}}, pk_{tag,c}\}_{sk_{tag,r}}$$

where $item_i = item$ in L_x, $data_i = pk_{tag,c}$, and

$$lic_{old,i} = tag = \{pk_{item}, L_x, data_{old} = pk_{tag,r}\}_{sk_{TGC}}$$

If the values $item_i$, $lic_{old,i}$, and $data_i$ have not been used before and the message is correctly signed with $sk_{tag,r}$, the tagged transaction simulator signs the message:

$$\{pk_{item}, L_x, data_i = pk_{tag,c}\}$$

using the key sk_{TGC} it generates and returns:

$$\{pk_{item}, L_x, data_i = pk_{tag,c}\}_{sk_{TGC}}$$

The value $L_x = \{item, License_x\}_{sk_{item}}$ is the same value that appears in $lic_{old,i}$.

Input:

The tagged transaction simulator provides input to the adversary that is computationally indistinguishable from the input it would receive in an actual run of the protocol. The adversary is given as input the values $item$, $data = pk$, pk_{item}, and pk_{TGC}. The tagged transaction simulator also gives the simulator access to the old license by constructing the input:

$$\{pk_{item}, L_x, data = pk\}_{sk_{TGC}}$$

This can be constructed because the tagged transaction simulator has access to the values pk_{item}, $data$, and the value $L_x = \{item, License_x\}_{sk_{item}}$ can be generated by the tagged transaction simulator as it has access to the values $item$ and sk_{item}.

Adversary:

Suppose adversary A passes the experiment with non negligible probability. Then the adversary has returned the value:

$$tag_A = \{pk_{item}, L_x, pk_r\}_{sk_{TGC}}$$

There are then two possible options: either the value tag_A was output from the O_{reg} oracle, or it was fabricated by the adversary. If the value:

$$tag_A = \{pk_{item}, L_x, pk_r\}_{sk_{TGC}}$$

was output from the O_{reg} oracle, then this adversary can be used to break the security of the signature scheme by completing an existential forgery using the key sk. Since the adversary is polynomially bounded, it can only make a maximum of n queries to the O_{reg} oracle where n is polynomially bounded. The tagged transaction simulator then goes through this list of input values it has received for the O_{reg} oracle. Either one of the values in the input will be:

$$\{tag_i = \{pk_{item}, L_x, pk\}_{sk_{TGC}}, pk_{tag,c}\}_{sk}$$

or the adversary has fabricated a tag which is detailed in the next paragraph. If the value:

$$\{tag_i = \{pk_{item}, L_x, pk\}_{sk_{TGC}}, pk_{tag,c}\}_{sk}$$

is in the input, then the tagged transaction simulator can use this as a valid message signature pair:

$$(\{tag_i, pk_{tag,c}\}, \{tag_i, pk_{tag,c}\}_{sk})$$

If the adversary has fabricated a license, the tagged transaction simulator re-runs the adversary but changes the input to $item$, $data = pk_{tag}$, pk_{item}, and $pk_{TGC} = pk$, where $item$, sk_{tag} and sk_{item} are randomly chosen and pk_{tag} and pk_{item} are generated from sk_{tag} and sk_{item}. The tagged transaction simulator also gives the simulator access to the old license by constructing the input:

$$\{pk_{item}, L_x, data = pk_{tag}\}_{sk}$$

using the signing oracle. The O_{reg} oracle is also changed. Rather than signing the message:

$$\{pk_{item}, L_x, data_i = pk_{tag,c}\}$$

itself, it will submit it to the signing oracle to sign and return:

$$\{pk_{item}, L_x, data_i = pk_{tag,c}\}_{sk}$$

This adversary is then used to break the security of the signature scheme by completing an existential forgery using the key $sk = sk_{TGC}$. As the adversary has fabricated a license, they return the value:

$$tag_A = \{pk_{item}, L_x, pk_r\}_{sk}$$

that has not been generated by a call to the O_{reg} query or provided as input. The tagged transaction simulator can use this message as a valid message signature pair:

$$(\{pk_{item}, L_x, pk_r\}, tag_A)$$

4.5.4 Cloning

In a cloning attack, the tagged transaction simulator is given as input the public key for the signature scheme pk and access to the signing oracle provided by the original simulator. The tagged transaction simulator is then challenged to break the security of the signature scheme by outputting a valid message signature pair. The tagged transactions simulator generates the values $item$, sk_{tag} and sk_{item} randomly using the random oracle of the original simulator. As $item$ is the output of a hash function it is computationally indistinguishable from a random value. The values sk_{item} and sk_{tag} are then used to generate the values pk_{item} and pk_{tag}.

Oracles:

The tagged transaction simulator implements the O_{reg} oracle. To simulate O_{reg} in a manner that is computationally indistinguishable from the real protocol, the tagged transaction simulator records in a list the input values it receives. The adversary will send the message:

$$\{tag = \{pk_{item}, L_x, pk_{tag,r}\}_{sk_{TGC}}, pk_{tag,c}\}_{sk_{tag,r}}$$

where $item_i = item$ in L_x, $data_i = pk_{tag,c}$, and

$$lic_{old,i} = \{pk_{item}, L_x, data_{old} = pk_{tag,r}\}_{sk_{TGC}}$$

If the values $item_i$, $lic_{old,i}$, and $data_i$ have not been used before and the message is correctly signed with $sk_{tag,r}$, the tagged transaction simulator submits the message:

$$\{pk_{item}, L_x, data_i = pk_{tag,c}\}$$

to the signing oracle to sign and returns:

$$\{pk_{item}, L_x, data_i = pk_{tag,c}\}_{sk_{TGC}}$$

The value $L_x = \{item, License_x\}_{sk_{item}}$ is the same value that appears in $lic_{old,i}$.

Input:

The tagged transaction simulator provides input to the adversary that is computationally indistinguishable from the input it would receive in an actual run of the protocol. The adversary is given as input the values $item$, $data = pk_{tag}$, pk_{item}, and $pk_{TGC} = pk$. The tagged transaction simulator also gives the simulator access to the old license by constructing the input:

$$tag = \{pk_{item}, L_x, data = pk_{tag}\}_{sk_{TGC}}$$

using the signing oracle as the tagged transaction simulator has access to the values pk_{item}, $data$, and the value $L_x = \{item, License_x\}_{sk_{item}}$.

Adversary:

Suppose adversary A passes the experiment with non negligible probability, then this adversary can be used to break the security of the signature scheme by completing an existential forgery using the key $sk_{TGC} = sk$. If the adversary has passed the experiment, it will have returned two tags:

$$tag_1 = \{pk_{item}, L_x, pk_{r1}\}_{sk_{TGC}}$$

and

$$tag_2 = \{pk_{item}, L_x, pk_{r2}\}_{sk_{TGC}}$$

There are three possible ways the adversary used the O_{reg} oracle to generate tag_1 and tag_2: the O_{reg} oracle output both tag_1 and tag_2, the O_{reg} oracle output one of tag_1 and tag_2, the O_{reg} oracle did not output either tag_1 or tag_2.

Since the adversary is polynomially bounded, it can only make a maximum of n queries to the O_{reg} oracle where n is polynomially bounded. If both tag_1 and tag_2 were output from the O_{reg} oracle, then the tagged transaction simulator can go through the list of n queries to the O_{reg} oracle to find the queries with the outputs of tag_1 and tag_2. The tagged transaction simulator can then repeat this

step finding the queries with the outputs that are the same as the previous steps' inputs. After a maximum of n steps, the tagged transaction simulator will have two lists of query inputs and outputs:

$$list_1 = (\{input_{1,1}, output_{1,1}\}, ..., \{input_{1,i}, tag_1\})$$

and

$$list_2 = (\{input_{2,1}, output_{2,1}\}, ..., \{input_{2,j}, tag_2\})$$

As the O_{reg} oracle will not generate two new tags for the same values of $item_i$, $lic_{old,i}$, and $data_i$, the values:

$$input_{1,1} = \{tag_{1,1} = \{\{pk_{item}, L_x, pk_{1,1}\}_{sk_{TGC}}, pk_{1,2}\}_{sk_{1,1}}$$

and

$$input_{2,1} = \{tag_{2,1} = \{\{pk_{item}, L_x, pk_{2,1}\}_{sk_{TGC}}, pk_{2,2}\}_{sk_{2,1}}$$

must be different. If $pk_{1,1} = data = pk_{tag}$, then the tagged transaction simulator uses:

$$(\{pk_{item}, L_x, pk_{2,1}\}, \{pk_{item}, L_x, pk_{2,1}\}_{sk_{TGC}})$$

as a valid message signature pair for the original simulator. If $pk_{2,1} = data = pk_{tag}$, then the tagged transaction simulator uses:

$$(\{pk_{item}, L_x, pk_{1,1}\}, \{pk_{item}, L_x, pk_{1,1}\}_{sk_{TGC}})$$

as a valid message signature pair for the original simulator.

If one (or both) of the values tag_1 and tag_2 was not in the output from the O_{reg} oracle, then the tagged transaction simulator can return a valid message signature pair to the original simulator. Suppose tag_1 was not in the output of the O_{reg} oracle, then the valid message signature pair is:

$$(\{pk_{item}, L_x, pk_{r1}\}, tag_1)$$

If tag_2 was not in the output of the O_{reg} oracle, then the valid message signature pair is:

$$(\{pk_{item}, L_x, pk_{r2}\}, tag_2)$$

4.5.5 Identity Revelation

To show that no message in the protocol reveals any information about the sender of the message, I construct a simulator that can simulate the messages that are sent in the protocol with no knowledge of the identity of any of the participants of the protocol. The simulator creates messages that are computationally indistinguishable from the messages in an actual run of the protocol. The simulator has access to the item x and the public variables pk_{TGC}, p, q, and g. The simulator also has access to a signing oracle for the TGC. This means the simulator can use the oracle to sign messages using the secret key sk_{TGC}. The TGC is not an anonymous party in the protocol so knowledge that the message was signed by the TGC does not reveal any information about the other parties in the protocol.

Many of the messages in the protocol are constructed by taking a random number x and raising a generator of the group g to the power of x to calculate $y = g^x \bmod p$. If the simulator and a real participant in the protocol both calculate these values (the simulator x_s and $y_s = g^{x_s} \bmod p$ and the actual participant x and $y = g^x \bmod p$), the values y_s and y are computationally indistinguishable.

$\{id = H(x), pk_x\}_{pk_{TGC}}$**:** The identity of the item $id = H(x)$ is a constant that can be constructed by the simulator as it has access to the item. The simulator then constructs the public key for the item by choosing a random value sk_x and calculating $pk_x = g^{-sk_x} \bmod p$.

$\{id = H(x), pk_{tag,r}\}$**:** The identity of the item $id = H(x)$ is a constant that can be constructed by the simulator as it has access to the item. The simulator will then construct the one time public key for the reseller and tag by choosing a random value $sk_{tag,r}$ and calculating $pk_{tag,r} = g^{-sk_{tag,r}} \bmod p$.

$\{L_x, pk_{tag,r}\}_{sk_x}$**:** The value $pk_{tag,r}$ has already been generated by the simulator. The value $L_x = \{id = H(x), License\}_{sk_x}$ is a signed license signed by the key sk_x already generated by the simulator. The value $License$ is a license for item, it may contain values such as the period of the license and other limitations. To prevent identity revelation the license must be constructed without using an identity information. If this is the case, the simulator can also generate the license.

$\{pk_x, L_x, pk_{tag,r}\}_{sk_{TGC}}$**:** The values pk_x, L_x, and $pk_{tag,r}$ have already been generated by the simulator.

$\{tag, pk_{tag,c}\}_{sk_{tag,r}}$**:** The values $sk_{tag,r}$ and tag have already been generated by the simulator. The simulator will then construct the one time public key for the customer and tag by choosing a random value $sk_{tag,c}$ and calculating $pk_{tag,c} =$

$g^{-sk_{tag,c}} \bmod p$.

4.5.6 Linkability

In any particular run of the protocol, a reseller or customer denoted r sends $\{id = H(x), pk_{tag,r}\}$ and (if the customer resells) $\{tag, pk_{tag,c}\}_{sk_{tag,r}}$. The values $id = H(x)$ and L_x are constant for every participant (assuming no identifying information in L_x). The variable values are $pk_{tag,r}$ and $pk_{tag,c}$. The public keys are computationally indistinguishable if the private keys are chosen randomly. If an adversary is able to link together the actions of a participant in separate runs of the protocol, then the adversary can distinguish between computationally indistinguishable values.

4.6 Modelling

To check the security properties of the tagged transaction protocol, I use the Failures Divergences Refinement (FDR) model checker [33, 78]. The FDR model checker checks a Concurrent Sequential Processes (CSP) [46] model of a protocol against a CSP specification. The FDR model checker was chosen because:

- It has been used extensively to analyse security protocols [29, 55, 57, 79, 81].
- It is supported by the Casper CSP compiler [56]. The Casper compiler takes a protocol description and compiles it to a CSP file that can be checked by FDR. The use of Casper makes it easier and quicker to construct CSP models of protocols and specifications although Casper is limited to compiling secret and authentication properties.

CSP models agents in the protocol as processes and messages are passed between processes over channels. The adversary is modelled by a process which starts with a set of initial knowledge. The adversary has the power to perform any action that a real world attacker could perform if it was in complete control of the network. These actions include:

- Intercept or overhear any messages sent over the network.
- Decrypt any messages that are encrypted with a key that the adversary knows.

- Fake new messages based on any information the adversary has or has learnt.
- Replay any message, even if the adversary cannot decrypt the message.

When analysing security protocols, a CSP model of the protocol is constructed and FDR checks that the model refines a CSP specification of the protocol. The FDR model checker is used to check safety properties. An example of a safety property for a security protocol is the failure of an adversary to discover a secret value. A CSP model of the protocol with the secret value is constructed, and the specification that states that the secret value should not be discovered by the adversary. The FDR model checker will then check all possible states of the protocol to see if the value is ever revealed to the adversary. When using a state based model checker to analyse security protocols, any encryption or digital signatures used are assumed to be perfectly secure. The FDR model checker enumerates all possible states and transitions between the states to check the specification against the model.

I have constructed three different models. One to represent the registration phase of the protocol, one to represent the supplier generating a tag, and one to represent a reseller generating a tag. The models do not examine anonymity or unlinkability and concentrate on the properties of spoofing, fabrication, cloning, and network sniffing. This chapter gives a brief description of the model of the protocol, followed by a brief description of the CSP specification. Appendix A shows the detailed CSP models and specifications of the tagged transaction protocol.

4.6.1 Safe Simplifying Transformations

The initial work on using CSP to verify security properties concentrated on small protocols like the Needham-Schroeder Public-Key Protocol [55]. Other security protocols are more complex, often involving more fields, messages, and layers of encryptions. Lowe et al have done work on safe simplifying transformations to remove some of these complexities and allow model checkers to check the security properties of more complex protocols without causing a state explosion [48]. The concept of a safe simplifying transformation is to apply a transformation to a protocol description to simplify the protocol while preserving any insecurities. If an attack exists on the original protocol then the attack still

exists on the simplified protocol.

Some examples of safe simplifying transformations include:

- Removing encryptions on fields in the messages.
- Removing hash functions on fields in the messages.
- Removing some atomic or hashed fields.
- Renaming atoms. An atomic field in a message can be renamed.
- Coalescing atoms. Two atomic fields in a message can be combined.

The transformations above are renaming of fields in messages within the protocol, there are also structural transformations that have been shown to be safe simplifications including:

- Splitting a message in two parts.
- Joining two messages into a single message.
- Redirecting a message that is sent via a third party so it is sent direct.

I make use of safe simplifying transformations to simplify the tagged transaction protocol before using the FDR model checker.

4.6.2 Registration

In the registration phase of the tagged transaction protocol, the supplier is registering the identity of a new item with the TGC along with a public key for the item. The message from the supplier to the TGC is encrypted with the public key for the TGC and the return message from the TGC is encrypted with the private key of the TGC. The high level description of the protocol is:

$Supplier \rightarrow TGC : \{item, itemkey\}_{pk_{TGC}}$
$TGC \rightarrow Supplier : \{item, itemkey\}_{sk_{TGC}}$

In this model, the identity of the item is represented as the set $Items = \{item\}$ and the keys for the item are represented by the set:

$$ItemKeys = \{itemkey, intruderkey\}$$

Each participant in the protocol has a public and private key pair. The initial knowledge of the adversary is $intruderkey$, pk_{TGC}, and $pk_{Supplier}$.

The registration step has to prevent a spoofing attack where an adversary registers the item before the supplier. The specification states that if the TGC receives a registration message for an item, the supplier must have sent the registration message. The FDR model checker returns TRUE after 10 states with 20 transitions.

4.6.3 Supplier Generating Tag

In the second phase of the tagged transaction protocol, the supplier generates a tag with the TGC. The reseller first sends the supplier the one time public key for the tag. The supplier then generates a signed license for the item. The supplier then sends a signed message with the license and the one time public key. The TGC then creates and signs the tag and sends it back to the supplier who sends it to the reseller. The high level description of the protocol is:

$Reseller \rightarrow Supplier : item, pk_{tag,r}$
$Supplier \rightarrow TGC : \{L_x = \{H(x), License_x\}_{sk_x}, pk_{tag,r}\}_{sk_x}$
$TGC \rightarrow Supplier : \{pk_x, L_x, pk_{tag,r}\}_{sk_{TGC}}$
$Supplier \rightarrow Reseller : \{pk_x, L_x, pk_{tag,r}\}_{sk_{TGC}}$

I apply the following safe simplifying transformations before modelling the supplier generating a tag with the TGC:

1. Removal of the encryption on the field $L_x = \{H(x), License_x\}_{sk_x}$ to make it $\{H(x), License\}$.
2. Removal of the atomic field $License$.
3. Removal of the atomic field $H(x)$.
4. Removal of the atomic field $item$.

After the simplification the description of the protocol is:

$Reseller \rightarrow Supplier : pk_{tag,r}$
$Supplier \rightarrow TGC : \{pk_{tag,r}\}_{sk_x}$
$TGC \rightarrow Supplier : \{pk_x, pk_{tag,r}\}_{sk_{TGC}}$

$Supplier \rightarrow Reseller : \{pk_x, pk_{tag,r}\}_{sk_{TGC}}$

In this model, the set $Tags = \{pktagreseller, pktagwrong\}$ represents the one time keys for the tags. As these keys are never used for encryption or signing in this part of the protocol they do not need to be represented as public and private key pairs. The value pk_x is represented as a public key with private key sk_x. Each participant in the protocol has a public and private key pair. The initial knowledge of the adversary is $pktagwrong$, pk_x, pk_{TGC}, $pk_{Supplier}$, $pk_{Reseller}$, and $sk_{Reseller}$. The model of the protocol for CSP is:

$Reseller \rightarrow Supplier : pktagreseller$
$Supplier \rightarrow TGC : \{pktagreseller\}_{sk_x}$
$TGC \rightarrow Supplier : \{pk_x, pktagreseller\}_{sk_{TGC}}$
$Supplier \rightarrow Reseller : \{pk_x, pktagreseller\}_{sk_{TGC}}$

The generation of a tag has to resist a fabrication attack where an adversary generates a valid tag. The specification states that if the reseller receives a tag that it accepts, the supplier must have sent a message to the TGC requesting the tag. The FDR model checker returns TRUE after 111 states with 387 transitions.

4.6.4 Reseller Generating Tag

In the third phase of the protocol, the reseller generates a tag with the TGC. The customer sends the one time key to the reseller who then forwards the one time key and the tag signed with the private key for the tag to the TGC. The TGC then checks that the tag has not been used before and generates a new tag that is sent back to the reseller and then back to the customer. This model checks whether the reseller can replay a tag as well as whether an adversary can network sniff a license. The high level description of the protocol is:

$Customer \rightarrow Reseller : item, pk_{tag,c}$
$Reseller \rightarrow TGC : \{\{pk_x, L_x, pk_{tag,r}\}_{sk_{TGC}}, pk_{tag,c}\}_{sk_{tag,r}}$
$TGC \rightarrow Reseller : \{pk_x, L_x, pk_{tag,c}\}_{sk_{TGC}}$
$Reseller \rightarrow Customer : \{pk_x, L_x, pk_{tag,c}\}_{sk_{TGC}}$

I apply the following safe simplifying transformations before modelling the

reseller generating a tag with the TGC:

1. Removal of the encryption on the field $L_x = \{H(x), License_x\}_{sk_x}$ to make it $\{H(x), License\}$.

2. Removal of the atomic field $License$.

3. Removal of the atomic field $H(x)$.

4. Removal of the atomic field $item$.

5. Redirecting a message sent via a third party to go direct. Message 3 is redirected to go straight from the TGC to the customer and not via the reseller.

This makes the high level description of the protocol after simplification:

$Customer \rightarrow Reseller : pk_{tag,c}$
$Reseller \rightarrow TGC : \{\{pk_x pk_{tag,r}\}_{sk_{TGC}}, pk_{tag,c}\}_{sk_{tag,r}}$
$TGC \rightarrow Customer : \{pk_x, pk_{tag,c}\}_{sk_{TGC}}$

I model the earlier generation of a tag ($\{pk_x, pk_{tag,r}\}_{sk_{TGC}}$) by the introduction of an extra set $ActualTags = \{tag, tag2, tagwrong, tagwrong2\}$ which represents the tag that has been sent from the supplier to the reseller. To represent the reseller signing the tag with the one time key embedded in the tag, the reseller signs the message with its own private key. Including the complete generation of a tag makes the model intractable. The set $Tags = \{pktag, pktag2, pktagwrong, pktagwrong2\}$ represents the one time keys for the new tags. The value pk_x is represented as a public key with private key sk_x. To model the possibility of a cloning attack, two runs of the protocol are run at the same time. The initial knowledge of the adversary is $tagwrong$, $tagwrong2$, $pktagwrong$, $pktagwrong2$, pk_x, pk_{TGC}, $pk_{Supplier}$, and $pk_{Reseller}$. The model of the protocol for CSP is:

$Customer \rightarrow Reseller : pktag, pktag2$
$Reseller \rightarrow TGC : \{tag, pktag, tag2, pktag2\}_{sk_R}$
$TGC \rightarrow Customer : \{pk_x, tag\}_{sk_{TGC}}, \{pk_x, tag2\}_{sk_{TGC}}$

There are two specifications for the reseller generating the tag. The first specification prevents cloning where an adversary creates two new tags from the same initial tag. The specification states that if the customer receives two tags, then two different values must have been signed by the TGC. The FDR model checker returns TRUE after 457 states with 2005 transitions. The second specification checks for a network sniffing attack where an adversary sees a tag being sent over the network and tries to use it to generate a new tag. The specification states that if a customer receives tags signed by the TGC then the reseller must have sent a message to the TGC to request the generation of the tags. The FDR model checker returns TRUE after 1423 states with 4835 transitions.

4.6.5 Remarks

Modelling a protocol using CSP can be used to show extra steps in the protocol. If the protocol is still secure when encryptions or signatures have been removed, then these extra operations are unnecessary to the security of the protocol. I have used this technique to refine the tagged transaction protocol.

In the initial design of the tagged transaction protocol a zero knowledge proof was used when the reseller generated a tag with the TGC. The reseller would use a zero knowledge proof of knowledge of a discrete logarithm to prove to the TGC that they knew the secret key that corresponded to the one time public key in the tag. The TGC could then use the property of witness extraction to discover the one time secret key if this tag was replayed. While modelling the protocol, I discovered that this step is unnecessary and the digital signature of the tag and new one time key are sufficient for the TGC to detect replay. The removing of the zero knowledge proof makes the tagged transaction protocol more efficient.

4.7 Summary

In this chapter, I have described and analysed the tagged transaction protocol. The tagged transaction protocol is fully described including the registering of items, the generation of tags by the supplier, and the subsequent generation of tags by resellers. The security analysis shows that the tagged transaction protocol is secure against spoofing, fabrication, cloning, network sniffing, identity revelation, and linkability attacks in the random oracle model. Finally, the details and results of the modelling of the tagged transaction protocol using the FDR

model checker show the protocol prevents spoofing, fabrication, cloning, and network sniffing attacks.

The tagged transaction protocol can prevent cloning attacks where an adversary tries to submit the same tag twice. In the next chapter, I describe an extended version of the tagged transaction protocol that not only detects cloning but is able to reveal the identity of the reseller that tried to clone the tag.

Chapter 5

Extended Tagged Transaction Protocol

This chapter describes an extended version of the tagged transaction protocol that provides revocable anonymity if a reseller tries to clone a tag. The extended tagged transaction protocol uses restricted blind signatures to provide revocable reseller anonymity while preserving anonymity and unlinkability. If a reseller uses a tag once, the reseller will remain anonymous and its independent transactions unlinkable. If the reseller tries to clone a tag by presenting the same tag to the TGC more than once, the identity of the reseller is revealed.

The extended tagged transaction protocol uses ideas from digital cash where a digital coin that is double spent reveals the identity of the double spender [18]. The TGCs acts as both the bank and the merchant. The reseller acts as the customer. The reseller will "withdraw" a coin from the TGC. This coin is then added to the tag that is signed by the TGC. As the TGC has blind signed the coin, it cannot link the signed coin to the reseller that withdrew it. When the reseller generates a new tag with the TGC, the reseller is "spending" the coin. If the coin is spent more than once, the identity of the reseller is revealed otherwise no information about the identity of the reseller is revealed.

The extended tagged transactions protocol uses the same protocols as the tagged transaction protocol with some modifications. All resellers and customers will have to register with the TGC. If the TGC detects a tag being cloned, it revokes this registration. Both customers and resellers will need to acquire identity tokens that have been blind signed by the TGC. Both the registration of resellers and customers and the acquisition of identity tokens are new protocols.

The protocols that involve the supplier and resellers generating tags with the TGC have also been modified to use identity tokens. When the supplier generates the tag with the TGC, they include an identity token from the reseller as well as the one time public key for this tag. When a reseller generates a tag with the TGC it takes part in a challenge-response protocol with the TGC to show that they own the identity token in the tag.

5.1 Restricted Blind Signatures

Chaum introduced the idea of blind signatures in 1982 [18]. This work was extended giving the first blind signature scheme based on RSA signatures [19]. In a blind signature scheme, the original message is blinded and the signer signs the blinded message. The blinding is then removed and the signature checked against the original message. Blind signatures were originally used in digital cash schemes to provide unlinkability and anonymity for the user when a bank blind signs a digital coin [20]. When blind signing a message, the signer does not know any information about the structure of the message being signed. To detect double spending of coins, the user encodes identifying information in the blinded digital coin to be signed by the bank in such a way that double spending the coin reveals their identity. In the paper by Chaum et al [20] they make use of a cut and choose technique to make sure the correct identifying information is encoded in a coin.

Restricted blind signatures were introduced by Stefan Brands [11]. In a restricted blind signature scheme, the signer can restrict the structure of the message being signed. In the withdrawal phase of the protocol, the bank inserts the public key of the user in the message to be signed. As this message is then blinded the bank can only discover the public key if it is spent twice. In digital cash schemes, this allows the signer to be sure that the user is correctly including information on their identity to detect double spending. The extended tagged transactions protocol also requires the encoding of identification information in the message to be blind signed.

Camenisch et al have produced a restricted blind signature scheme called CL-signatures [16]. These restricted signatures have been used in digital cash [14] and anonymous credential schemes [13, 15]. CL-signatures can be used to sign a committed value and to prove knowledge of a signature using zero knowledge

proofs.

In the extended tagged transaction protocol, the TGC acts as both the bank and the merchant while the resellers and customers act as users. To generate restricted blind signed values to use in the tags, customers and resellers take part in a protocol with the TGC. This protocol can be done in advance. For example, a customer or reseller may generate ten restricted blind signed values before taking part in a transaction. For the extended tagged transaction protocol, the restricted blind signature scheme must be efficient at verifying the signatures. The restricted blind signature scheme by Brands [11] requires fewer operations to verify the signature than CL-signatures [16] because Cl-signatures use zero knowledge proofs to show the value was signed by the TGC.

Recently, Henry et al have introduced the notion of verifier efficient restricted blind signatures (VERBS) [45]. VERBS require no modular exponentiation to verify the signature. In the security analysis of VERBS, the authors do not show the security of the restricted blind signature scheme equivalent to a known hard problem but do discuss the security of VERBS. In the signature scheme of Brands, the security is shown to be equivalent to solving the representation problem in groups of prime order. For this reason, I make use of the restricted signature scheme by Brands, but the VERBS signature scheme could also be used in the extended tagged transaction protocol.

5.2 Definitions

The following notation is used to denote restrictive blind signatures:

- ID_r the unique identity of the participant r. This corresponds to the account number in digital cash. It is registered with the TGC and is revealed if a tag is cloned.

- pk_r the public key of participant r. This is used to sign identity token requests to prevent an adversary from being able to generate tokens for another party.

- $token_{tag,r}$ an identity token for participant r and tag. On its own it does not reveal any information about participant r.

- $token'_{tag,r}$ a blinded identity token for participant r and tag.

- pk_{ID} and sk_{ID} are the public and secret key of the TGC for blind signing identity tokens. The value sk_{ID} is known only to the TGC. The value pk_{ID} is a well known public value.

In the extended tagged transaction protocol we add a new parameter to the tag definition. The new parameter is a signature from the TGC on an identity token that can be used to reveal the identity of a reseller that tries to clone a tag. An extended tag is a tuple:

$$tag = \{A = pk_x, B = L_x, C = \{token_{tag,r}\}_{sk_{ID}}, D = pk_{tag,r}\}$$

The definitions of the parameters in the tag are:

- $A = pk_x$: the public key for the item. This is registered with the TGC when the supplier first creates the item. The TGC should only sign tags that have the correct public key for the supplier.

- $B = L_x = \{id = H(x), License\}_{sk_x}$: a license signed with the secret key for the item x. This license contains information such as the identity of the item $id = H(x)$ and any other important terms of the license. The structure of licenses is discussed in Section 4.1.2. The identity of the item $id = H(x)$ is calculated using a well known hash function H. Identifying items is discussed in Section 4.1.1.

- $C = \{token_{tag,r}\}_{sk_{ID}}$. The identity token for participant r and tag. The identity token is signed using the secret key of the TGC for blind signatures sk_{ID}. The identity token does not reveal any information about the participant unless the tag is cloned.

- $D = pk_{tag,r}$: the one time public key for the participant r and tag tag. The reseller or customer randomly chooses $sk_{tag,r}$ and uses it to generate the public key.

5.3 Registering with the TGC

Initially customers and resellers will need to register with the TGC. This is the same as opening a bank account in the digital cash scenario. This only needs to be done once by every customer and reseller.

The process for resellers and customers to register with the TGC takes place in two steps as shown in Figure 5.1.

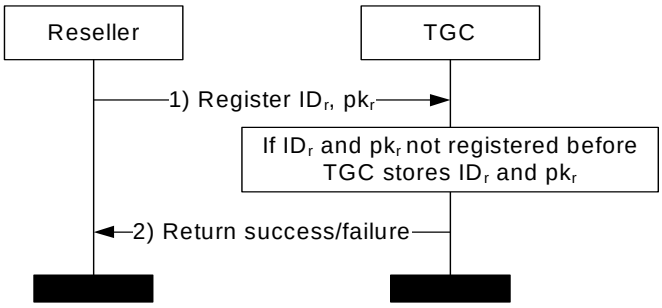

Figure 5.1: Registering with the TGC

1. The reseller or customer chooses an identity ID_r and public key pk_r and sends these to the TGC.

2. The TGC checks whether the identity ID_r and public key pk_r have been used before. If the identity and public key are unique, the TGC will store these values and return a success message. If either the identity ID_r or the public key pk_r are not unique, the TGC returns a failure message and the reseller or customer generates new values and tries again.

5.4 Generating ID Token

Before a customer or reseller can request a tag, they must generate an identity token signed by the TGC. The TGC will blind sign the identity token submitted to them. The blind signature prevents the TGC from being able to link this identity token to the reseller when the identity token is used at a later time to generate a tag. A restrictive blind signature scheme is used so that customers and resellers can only obtain identity tokens for identities for whom they know the secret values.

The process for a reseller to generate a blind signed identity token with the TGC takes place in five steps as shown in Figure 5.2.

1. The reseller generates a request for a signed identity token and authenticates itself with the TGC by signing the message with its private key. The TGC can look up the identity that corresponds to this public key.

2. The reseller generates a blinded identity token.

3. The reseller sends this blinded token item to the TGC.

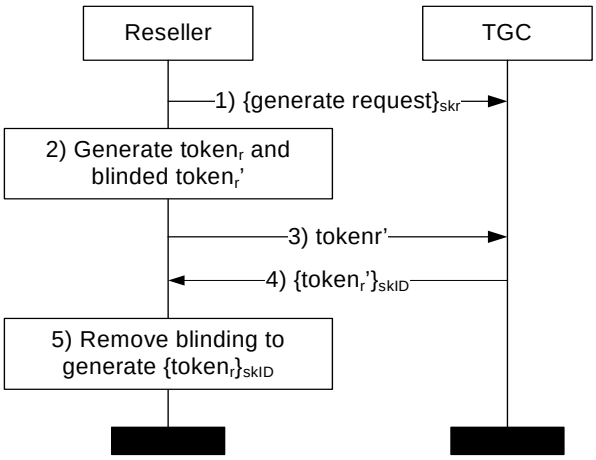

Figure 5.2: Generating ID Token

4. The TGC blind signs the identity token.

5. The reseller removes the blinding on the signed identity token.

Due to the restricted blind signature scheme used it should not be possible for the reseller to have a signed identity token that was not signed by the TGC. The restricted nature of the blind signature scheme also prevents the reseller from getting an identity token that does not contain information on the identity it has registered with the TGC. The final signed identity token does not reveal any information about the identity of the reseller.

It is worth noting that while the final signed identity token does not reveal any information about the customer or reseller, the TGC learns how many identity tokens have been generated by each customer and reseller. This information could be used to infer how many transactions each customer and reseller is taking part in, although there does not need to be a one-to-one relationship between the number of identity tokens generated and the number of transactions.

5.5 Supplier Generating Tag with TGC

To generate a tag for an item, a protocol takes place between the reseller, the supplier, and the TGC. The communication between the supplier and the TGC is done over an anonymous channel, if supplier anonymity is required, as shown by dotted lines in Figure 5.3. Compared to the tagged transaction protocol, the only differences are the inclusion of the signed identity token in the tag. The process

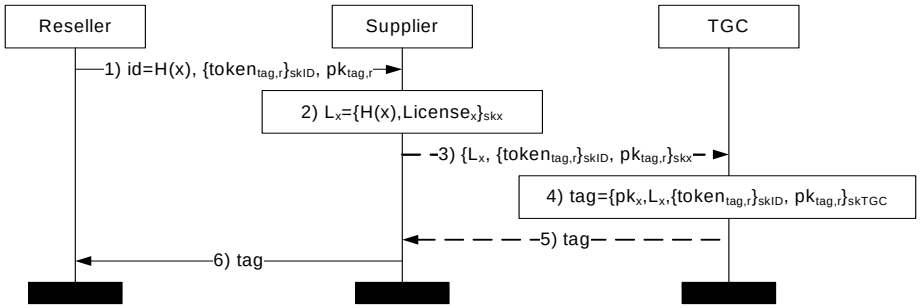

Figure 5.3: Supplier Generating Tag with TGC

for a supplier to generate a tag with the TGC takes part in six steps as shown in Figure 5.3.

1. The reseller sends a purchase request to the supplier containing the identity of the item they wish to purchase $id = H(x)$, the signed identity token $\{token_{tag,r}\}_{sk_{ID}}$, and the one time public key for the tag $pk_{tag,r}$. The reseller randomly picks a private key and uses this to generate the public key.

2. The supplier signs a license for the item $L_x = \{H(x), License_x\}_{sk_x}$ where $H(x)$ is the identity of the item and $License_x$ is the license for the item.

3. The supplier creates a signed tag request containing the license L_x, the signed identity token $\{token_{tag,r}\}_{sk_{ID}}$, and the one time public key $pk_{tag,r}$ all signed by sk_x and sends it to the TGC.

4. The TGC checks the one time public key $pk_{tag,r}$ and the signed identity token $\{token_{tag,r}\}_{sk_{ID}}$ have not been used for this item before, that the identity token has been signed using sk_{ID}, and that the key is correct for this item. If these tests pass the TGC constructs and signs $tag = \{pk_x, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}_{sk_{TGC}}$.

5. The TGC sends the tag to the supplier.

6. The supplier sends the tag to the reseller. The reseller checks the tag has been signed by the TGC, that the license is for the correct item, and that the tag contains the correct signed identity token and one time public key.

The supplier signs the license generation request with the secret key for the item to prevent any other party from being able to generate tags for the item with

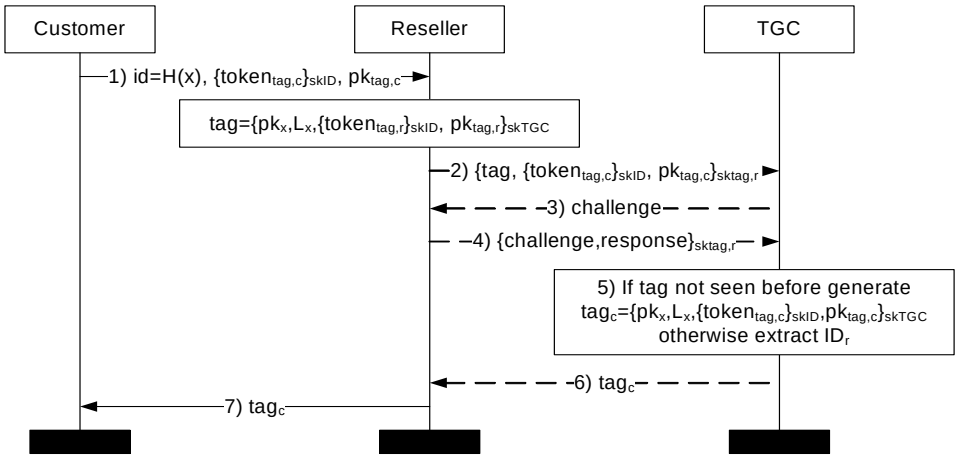

Figure 5.4: Reseller Generating Tag with TGC

the TGC. The reseller will only accept the tag if the tag is signed by the TGC and contains the correct signed identity token and one time key.

5.6 Reseller Generating Tag with TGC

Now the tag for an item has been generated, signed, and sent to the reseller, the reseller can use this tag to generate a new tag for a customer or another reseller without input from the supplier. The generation of a new tag is shown in Figure 5.4. The communication between the reseller and the TGC is done over an anonymous channel as shown by dotted lines. Compared to the tagged transaction protocol, the extended tagged transaction protocol includes a challenge response protocol on the signed identity token. If the tag has not been cloned, no information is revealed about the identity of the reseller. If the tag is cloned, the identity of the reseller is revealed by the challenge response protocol.

The generation of a new tag by a reseller takes place in the seven steps shown in Figure 5.4.

1. The customer sends a purchase request comprised of the identity of the item they wish to purchase, the signed identity token $\{token_{tag,c}\}_{sk_{ID}}$, and the one time public key for the tag $pk_{tag,c}$. The customer randomly picks a private key and uses this to generate the public key.

2. The reseller sends a message with the signed identity token $\{token_{tag,c}\}_{sk_{ID}}$, the one time public key chosen by the customer $pk_{tag,c}$, and the tag signed

using the one time secret key used in the generation of the tag $sk_{tag,r}$ to the TGC.

3. The TGC sends the reseller a challenge value $challenge$.

4. The reseller sends the TGC a response value in a signed message $\{challenge, response\}_{sk_{tag,r}}$. The reseller can only provide the correct response value if they know the secret value associated with the identity value.

5. If the tag has not been cloned the TGC generates a new tag

$$tag_c = \{pk_x, L, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk_{TGC}}$$

and saves the values $\{token_{tag,r}\}_{sk_{ID}}$ and $\{challenge, response\}_{sk_{tag,r}}$. If the tag has been cloned, then the TGC will have saved the values $\{token_{tag,r}\}_{sk_{ID}}$ and $\{challenge_2, response_2\}_{sk_{tag,r}}$ from the previous run. Using the value $\{token_{tag,r}\}_{sk_{ID}}$ and the two challenge and response values $challenge$, $challenge_2$, $response$, and $response_2$ the TGC extracts the identity of the reseller ID_r. The TGC presents the values ID_r, $\{token_{tag,r}\}_{sk_{ID}}$, $\{challenge, response\}_{sk_{tag,r}}$, and $\{challenge_2, response_2\}_{sk_{tag,r}}$ as proof the tag has been cloned.

6. The TGC sends tag_c to the reseller.

7. The reseller sends tag_c to the customer. The customer only accepts the tag if the tag is signed by the TGC and contains the correct signed identity token and one time key.

In step 5 of the extended tagged transaction protocol, the identity of a reseller that clones a tag is revealed. To reveal the identity of a reseller who clones a tag, a challenge response protocol is used. The restricted blind signature $\{token_{tag,r}\}_{sk_{ID}}$ contains a commitment value. A challenge response protocol takes place between the reseller and the TGC where the reseller shows the TGC that they know a secret value embedded in the blind signature. When a challenge response protocol is executed with the same commitment value and different challenge and response values the secret value is revealed. If a challenge response protocol takes place with a commitment value and only one challenge and response value then no information about the secret value is revealed. This is a property of the restricted blind signature scheme used. Blind signatures (and restricted blind signatures) use this property to detect double spending when applied to digital cash.

5.7 Security Analysis

Thereom 2. *The extended tagged transaction protocol provides Secure Provenance (Section 3.5.7) in the random oracle model provided that the signature scheme used has provable security against existential forgeries under adaptive chosen message attacks, the encryption scheme used has provable security against IND-CCA2 attacks, and the restrictive blind signature scheme provides anonymity and unlinkability.*

I use a reduction to contradiction style of argument to show the extended tagged transaction protocol provides security against fabrication, network sniffing, and cloning attacks. The security argument for the prevention of spoofing is the same as for the original tagged transaction protocol shown in Section 4.5. In this security analysis, the TGC is assumed to be acting as a trusted third party. Chapter 6 removes this assumption and discusses methods to verify the actions of the TGC. For the security analysis of the identity revelation and linkability properties the following assumptions are made: a perfect anonymous communication channel, an anonymous supplier, and the parties in the protocol not revealing their identities or the identities of the parties they are communicating with. This security analysis does not consider side channel attacks.

The tagged transaction simulator provides the tagged transaction adversary with inputs that are computationally indistinguishable from the inputs it would receive in an actual run of the protocol. If the tagged transaction adversary breaks the security of the extended tagged transaction protocol in this simulated run, then the tagged transaction simulator can provide an output that can solve a problem thought to be hard.

The tagged transaction simulator breaks the security of the signature scheme if, when given public key pk and access to a signing oracle provided by the original simulator, the tagged transaction simulator outputs a valid message signature pair (m, σ) with non negligible probability where m was not used in a query to the signing oracle.

The tagged transaction simulator will need to be able to simulate the generation of blind signed ID tokens that are computationally indistinguishable from the ID tokens generated in an actual run of the protocol. The tagged transaction simulator initially generates a random private key $sk_{ID} = x$ and public keys $pk_{ID} = \{g, g_1, g_2\}$ for the restricted blind signature scheme. To register a reseller, the adversary will send the message I to the tagged transaction simulator. The tagged transaction simulator will check that $Ig_2 \neq 1$ and generate and return

$z = (Ig_2)^x$. As x is chosen randomly this is computationally indistinguishable from an actual run of the protocol. To generate an id token, the tagged transaction simulator will send the adversary the messages $a = g^w$ and $b = (Ig_2)^w$ where w is chosen randomly. When the adversary sends the challenge message c, the tagged transaction simulator returns the message $r = cx + w \bmod q$. As w is chosen randomly this is computationally indistinguishable from an actual run of the protocol.

When providing inputs to the tagged transaction adversary and analysing the identity revelation and unlinkability properties, the security arguments involve showing computational indistinguishability of elements in a group. For a cyclic group $\mathbb{G}$ of order q with generator g, when given an element z chosen at random from $\mathbb{Z}_q$, the element g^z is uniformly distributed in $\mathbb{G}$. Two elements uniformly distributed in $\mathbb{G}$ are computationally indistinguishable. If two elements a_1 and a_2 are chosen at random from $\mathbb{Z}_q$, the elements g^{a_1} and g^{a_2} are computationally indistinguishable. If two private keys for the Schnorr signature scheme are chosen at random, the public keys are computationally indistinguishable. A random element from $\mathbb{G}$ and a Schnorr public key are computationally indistinguishable. Computational indistinguishability is preserved under efficient (Probabilistic Polynomial Time) operations.

5.7.1 Fabrication

In a fabrication attack, the tagged transaction simulator is given as input the public key for the signature scheme pk and access to the signing oracle provided by the original simulator. The tagged transaction simulator is then challenged to break the security of the signature scheme by outputting a valid message signature pair. The tagged transactions simulator generates the values $item$, pk_r, and sk_{item} randomly using the random oracle of the original simulator. As $item$ is the output of a hash function it is computationally indistinguishable from a random value. The value sk_{item} is then used to generate the value pk_{item}.

Oracles:

The tagged transaction simulator implements the O_{gen} and O_{reg} oracles. To simulate O_{gen} in a manner that is computationally indistinguishable from the real protocol, the tagged transaction simulator records in a list the input values it receives. The adversary will send the message:

$$\{id, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}$$

where $item_i = id$ and $data_i = pk_{tag,r}$. If the values $item_i$ and $data_i$ have not been used before and the value $\{token_{tag,r}\}_{sk_{ID}}$ is correctly signed by the key sk_{ID}, the tagged transaction simulator submits the message

$$m = \{pk_{item}, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}$$

to the signing oracle and returns

$$tag = \{pk_{item}, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}_{sk}$$

The value $L_x = \{item, License_x\}_{sk_{item}}$ can be generated by the tagged transaction simulator as it has access to the values $item$ and sk_{item}.

To simulate O_{reg} in a manner that is computationally indistinguishable from the real protocol, the tagged transaction simulator records in a list the input values it receives. The adversary will send the message:

$$\{\{pk_{item}, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}_{sk_{TGC}}, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk_{tag,r}}$$

where $item_j = item$ in L_x, $data_j = pk_{tag,c}$, and

$$lic_{old,j} = \{pk_{item}, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}_{sk_{TGC}}$$

The tagged transaction simulator checks the message is correctly signed with $sk_{tag,r}$ and that $\{token_{tag,r}\}_{sk_{ID}}$ and $\{token_{tag,c}\}_{sk_{ID}}$ have been correctly signed by the key sk_{ID}. The tagged transaction simulator returns the challenge value d to the adversary and waits for the adversary to reply with the responses r_1 and r_2. If the response values are correct and the values $item_j$, $lic_{old,j}$, and $data_j$ have not been used before, the tagged transaction simulator submits the message

$$\{pk_{item}, L_x, \{token_{tag,c}\}_{sk_{ID}}, data_j = pk_{tag,c}\}$$

to the signing oracle and returns

$$\{pk_{item}, L_x, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk_{TGC}}$$

The value $L_x = \{item, License_x\}_{sk_{item}}$ is the same value that appears in $lic_{old,j}$.

Input:

The tagged transaction simulator provides input to the adversary that is computationally indistinguishable from the input it would receive in an actual run of the protocol. The adversary is given as input the values $item$, $data = pk_r$, pk_{item}, and $pk_{TGC} = pk$.

Adversary:

Suppose adversary A passes the experiment with non negligible probability. The adversary is then able to fabricate a valid tag:

$$tag = \{pk_{item}, L_x, \{token_r\}_{sk_{ID}}, pk_r\}_{sk}$$

with non negligible probability. The simulator returns this to the original simulator as it has a valid message signature pair

$$(\{pk_{item}, L_x, \{token_r\}_{sk_{ID}}, pk_r\}, tag)$$

The value

$$\{pk_{item}, L_x, \{token_r\}_{sk_{ID}}, pk_r\}$$

will not have been used in a query to the signing oracle as neither the O_{gen} or O_{reg} oracles can be used with the value $data = pk_r$.

5.7.2 Network Sniffing

In a network sniffing attack, the tagged transaction simulator is given as input the public key for the signature scheme pk and access to the signing oracle provided by the original simulator. The tagged transaction simulator is then challenged to break the security of the signature scheme by outputting a valid message signature pair. The tagged transactions simulator generates the values $item$, sk_{TGC} and sk_{item} randomly using the random oracle of the original simulator. As $item$ is the output of a hash function it is computationally indistinguishable from a random value. The values sk_{item} and sk_{TGC} are then used to generate the values pk_{item} and pk_{TGC}.

Oracles:

The tagged transaction simulator implements the O_{reg} oracle. To simulate O_{reg} in a manner that is computationally indistinguishable from the real protocol, the tagged transaction simulator records in a list the input values it receives. The adversary will send the message:

$$\{tag = \{pk_{item}, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}_{sk_{TGC}}, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk_{tag,r}}$$

where $item_i = item$ in L_x, $data_i = pk_{tag,c}$, and

$$lic_{old,i} = tag = \{pk_{item}, L_x, \{token_{tag,r}\}_{sk_{ID}}, data_{old} = pk_{tag,r}\}_{sk_{TGC}}$$

The tagged transaction simulator checks the message is correctly signed with $sk_{tag,r}$ and that $\{token_{tag,r}\}_{sk_{ID}}$ and $\{token_{tag,c}\}_{sk_{ID}}$ have been correctly signed by the key sk_{ID}. The tagged transaction simulator then returns the challenge value d to the adversary and waits for the adversary to reply with the responses r_1 and r_2. If the response values are correct and the values $item_j$, $lic_{old,j}$, and $data_j$ have not been used before, the tagged transaction simulator signs the message

$$\{pk_{item}, L_x, \{token_{tag,c}\}_{sk_{ID}}, data_i = pk_{tag,c}\}$$

using the key sk_{TGC} it generated and returns

$$\{pk_{item}, L_x, \{token_{tag,c}\}_{sk_{ID}}, data_i = pk_{tag,c}\}_{sk_{TGC}}$$

The value $L_x = \{item, License_x\}_{sk_{item}}$ is the same value that appears in $lic_{old,i}$.

Input:

The tagged transaction simulator provides input to the adversary that is computationally indistinguishable from the input it would receive in an actual run of the protocol. The adversary is given as input the values $item$, $data = pk$, pk_{item}, and pk_{TGC}. The tagged transaction simulator also gives the simulator access to the old license by constructing the input:

$$\{pk_{item}, L_x, \{token\}_{sk_{ID}}, data = pk\}_{sk_{TGC}}$$

as the tagged transaction simulator has access to the values pk_{item}, $data$, and the value $L_x = \{item, License_x\}_{sk_{item}}$ can be generated by the tagged transaction simulator as it has access to the values $item$ and sk_{item}. The tagged transaction simulator will also need to generate the value $\{token\}_{sk_{ID}}$. The tagged transaction simulator can generate this value as it has access to the secret key sk_{ID} and can generate $token$ for a random user. This is computationally indistinguishable from an actual run of the protocol as the actual user will have chosen the value in $token$ randomly.

Adversary:

Suppose adversary A passes the experiment with non negligible probability. Then the adversary has returned the value

$$tag_A = \{pk_{item}, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_r\}_{sk_{TGC}}$$

There are then two possible options: either the value tag_A was output from the O_{reg} oracle, or it was fabricated by the adversary.

If the value

$$tag_A = \{pk_{item}, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_r\}_{sk_{TGC}}$$

was output from the O_{reg} oracle, then this adversary can be used to break the security of the signature scheme by completing an existential forgery using the key sk. Since the adversary is polynomially bounded, it can only make a maximum of n queries to the O_{reg} oracle where n is polynomially bounded. The tagged transaction simulator then goes through this list of input values it has received for the O_{reg} oracle. Either one of the values in the input will be:

$$\{tag_i = \{pk_{item}, L_x, \{token\}_{sk_{ID}}, pk\}_{sk_{TGC}}, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk}$$

or the adversary has fabricated a tag which is the same as the case detailed in the next paragraph. If the value

$$\{tag_i = \{pk_{item}, L_x, \{token\}_{sk_{ID}}, pk\}_{sk_{TGC}}, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk}$$

is in the input, then the tagged transaction simulator can construct a valid message signature pair

$$(\{tag_i, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}, \{tag_i, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk})$$

If the adversary has fabricated a license, the tagged transaction simulator re-runs the adversary but changes the input to $item$, $data = pk_{tag}$, pk_{item}, and $pk_{TGC} = pk$, where $item$, sk_{tag} and sk_{item} are randomly chosen and pk_{tag} and pk_{item} are generated from sk_{tag} and sk_{item}. The tagged transaction simulator also gives the simulator access to the old license by constructing the input

$$\{pk_{item}, L_x, \{token\}_{sk_{ID}}, data = pk_{tag}\}_{sk}$$

using the signing oracle. The O_{reg} oracle is also changed. Rather than signing the message

$$\{pk_{item}, L_x, \{token\}_{sk_{ID}}, data_i = pk_{tag,c}\}$$

itself, it will submit it to the signing oracle to sign and return

$$\{pk_{item}, L_x, \{token_{tag,c}\}_{sk_{ID}}, data_i = pk_{tag,c}\}_{sk}$$

This adversary is then used to break the security of the signature scheme by completing an existential forgery using the key $sk = sk_{TGC}$. As the adversary has fabricated a license, they return the value

$$tag_A = \{pk_{item}, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_r\}_{sk}$$

that has not been generated by a call to the O_{reg} query or provided as input. The tagged transaction simulator can use this message as a valid message signature pair

$$(\{pk_{item}, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_r\}, tag_A)$$

5.7.3 Cloning

In a cloning attack, the tagged transaction simulator is given as input the public key for the signature scheme pk and access to the signing oracle provided by the original simulator. The tagged transaction simulator is then challenged to break the security of the signature scheme by outputting a valid message signature pair. The tagged transactions simulator generates the values $item$, sk_{tag} and sk_{item} randomly using the random oracle of the original simulator. As $item$ is the output of a hash function it is computationally indistinguishable from a random value. The values sk_{item} and sk_{tag} are then used to generate the values pk_{item} and pk_{tag}.

Oracles:

The tagged transaction simulator implements the O_{reg} oracle. To simulate O_{reg} in a manner that is computationally indistinguishable from the real protocol, the tagged transaction simulator records in a list the input values it receives. The adversary will send the message:

$$\{tag = \{pk_{item}, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}_{sk_{TGC}}, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk_{tag,r}}$$

where $item_i = item$ in L_x, $data_i = pk_{tag,c}$, and

$$lic_{old,i} = \{pk_{item}, L_x, \{token_{tag,r}\}_{sk_{ID}}, data_{old} = pk_{tag,r}\}_{sk_{TGC}}$$

The tagged transaction simulator checks the message is correctly signed with $sk_{tag,r}$ and that $\{token_{tag,r}\}_{sk_{ID}}$ and $\{token_{tag,c}\}_{sk_{ID}}$ have been correctly signed by the key sk_{ID}. The tagged transaction simulator then returns the challenge value d to the adversary and waits for the adversary to reply with the responses r_1 and r_2. If the response values are correct and the values $item_j$, $lic_{old,j}$, and $data_j$ have not been used before, the tagged transaction simulator submits the message

$$\{pk_{item}, L_x, \{token_{tag,c}\}_{sk_{ID}}, data_i = pk_{tag,c}\}$$

to the signing oracle to sign and returns

$$\{pk_{item}, L_x, \{token_{tag,c}\}_{sk_{ID}}, data_i = pk_{tag,c}\}_{sk}$$

The value $L_x = \{item, License_x\}_{sk_{item}}$ is the same value that appears in $lic_{old,i}$.

Input:

The tagged transaction simulator provides input to the adversary that is computationally indistinguishable from the input it would receive in an actual run of the protocol. The adversary is given as input the values $item$, $data = pk_{tag}$, pk_{item}, and $pk_{TGC} = pk$. The tagged transaction simulator also gives the simulator access to the old license by constructing the input

$$tag = \{pk_{item}, L_x, \{token_{tag}\}_{sk_{ID}}, data = pk_{tag}\}_{sk}$$

using the signing oracle as the tagged transaction simulator has access to the values pk_{item}, $data$, and the value $L_x = \{item, License_x\}_{sk_{item}}$. To generate the value $\{token_{tag}\}_{sk_{ID}}$ the tagged transaction simulator will need access to the identifier for the adversary I and the secret key sk_{ID}. As the only method for the adversary to register for a token is through the tagged transaction simulator, the tagged transaction simulator has access to both I and sk_{ID}.

Adversary:

Suppose adversary A passes the experiment with non negligible probability, then this adversary can be used to break the security of the signature scheme by completing an existential forgery using the key $sk_{TGC} = sk$. If the adversary has passed the experiment, it will have returned two licenses:

$$tag_1 = \{pk_{item}, L_x, \{token_{r1}\}_{sk_{ID}}, pk_{r1}\}_{sk}$$

and

$$tag_2 = \{pk_{item}, L_x, \{token_{r2}\}_{sk_{ID}}, pk_{r2}\}_{sk}$$

There are three possible ways the adversary used the O_{reg} oracle to generate tag_1 and tag_2: the O_{reg} oracle output both tag_1 and tag_2, the O_{reg} oracle output one of tag_1 and tag_2, the O_{reg} oracle did not output either tag_1 or tag_2.

Since the adversary is polynomially bounded, it can only make a maximum of n queries to the O_{reg} oracle where n is polynomially bounded. If both tag_1 and tag_2 were output from the O_{reg} oracle, then the tagged transaction simulator can go through the list of n queries to the O_{reg} oracle to find the queries with the outputs of tag_1 and tag_2. The tagged transaction simulator can then repeat this step finding the queries with the outputs that are the same as the previous steps inputs. After a maximum of n steps, the tagged transaction simulator will have two lists of query inputs and outputs:

$$list_1 = (\{input_{1,1}, output_{1,1}\}, ..., \{input_{1,i}, tag_1\})$$

and

$$list_2 = (\{input_{2,1}, output_{2,1}\}, ..., \{input_{2,j}, tag_2\})$$

As the O_{reg} oracle will not generate two new tags for the same values of $item_i$, $lic_{old,i}$, and $data_i$, the values

$$input_{1,1} = \{\{pk_{item}, L_x, \{token_{1,1}\}_{sk_{ID}}, pk_{1,1}\}_{sk_{TGC}}, \{token_{1,2}\}_{sk_{ID}}, pk_{1,2}\}_{sk_{1,1}}$$

and

$$input_{2,1} = \{\{pk_{item}, L_x, \{token_{2,1}\}_{sk_{ID}}, pk_{2,1}\}_{sk_{TGC}}, \{token_{2,2}\}_{sk_{ID}}, pk_{2,2}\}_{sk_{2,1}}$$

must be different. If $pk_{1,1} = data = pk_{tag}$, then the tagged transaction simulator uses

$$(\{pk_{item}, L_x, \{token_{2,1}\}_{sk_{ID}}, pk_{2,1}\}, \{pk_{item}, L_x, \{token_{2,1}\}_{sk_{ID}}, pk_{2,1}\}_{sk})$$

as a valid message signature pair for the original simulator. If $pk_{2,1} = data = pk_{tag}$, then the tagged transaction simulator uses

$$(\{pk_{item}, L_x, \{token_{1,1}\}_{sk_{ID}}, pk_{1,1}\}, \{pk_{item}, L_x, \{token_{1,1}\}_{sk_{ID}}, pk_{1,1}\}_{sk})$$

as a valid message signature pair for the original simulator.

If one (or both) of the values tag_1 and tag_2 was not in the output from the O_{reg} oracle, then the tagged transaction simulator can return a valid message signature pair to the original simulator. Suppose tag_1 was not in the output of the O_{reg} oracle, then the valid message signature pair is

$$(\{pk_{item}, L_x, \{token_{r1}\}_{sk_{ID}}, pk_{r1}\}, tag_1)$$

If tag_2 was not in the output of the O_{reg} oracle, then the valid message signature pair is

$$(\{pk_{item}, L_x, \{token_{r2}\}_{sk_{ID}}, pk_{r2}\}, tag_2)$$

5.7.4 Identity Revelation

To show that no message in the protocol reveals any information about the sender, I construct a simulator that can simulate the messages that are sent in the protocol with no knowledge of the identity of any of the participants of the protocol. The simulator creates messages that are computationally indistinguishable from the messages in an actual run of the protocol. The simulator has access to the item x,

the public variables of the system pk_{TGC}, p, q, and g and the public keys $\{g, g_1, g_2\}$ of the restricted blind signature scheme.

The simulator also has access to signing oracles for the TGC. This means the simulator can use the oracles to sign messages using the secret keys sk_{TGC} and sk_{ID}. The TGC is not an anonymous party in the protocol so knowledge that the message was signed by the TGC does not reveal any information about the other parties in the protocol.

Many of the messages in the protocol are constructed by taking a random number x and raising a generator of the group g to the power of x to calculate $y = g^x \bmod p$. If the simulator and a real participant in the protocol both calculate these values (the simulator x_s and $y_s = g^{x_s} \bmod p$ and the actual participant x and $y = g^x \bmod p$), the values y_s and y are computationally indistinguishable.

$\{id = H(x), pk_x\}_{pk_{TGC}}$**:** The identity of the item $id = H(x)$ is a constant that can be constructed by the simulator as it has access to the item. The simulator then constructs the public key for the item by choosing a random value sk_x and calculating $pk_x = g^{-sk_x} \bmod p$.

$\{id = H(x), \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}$**:** The identity of the item $id = H(x)$ is a constant that can be constructed by the simulator as it has access to the item. The value $\{token_{tag,r}\}_{sk_{ID}}$ consists of the value $token$ which is generated using an identity value I_s which the simulator generates at random. This identity value is computationally indistinguishable from the identity value I that is randomly chosen in an actual run of the protocol. All other values chosen to compute $\{token_{tag,r}\}_{sk_{ID}}$ are products of random values that are computationally indistinguishable from an actual run of the protocol. The simulator can then use the signing oracle to sign the message $token_{tag,r}$. The simulator will then construct the one time public key for the reseller and tag by choosing a random value $sk_{tag,r}$ and calculating $pk_{tag,r} = g^{-sk_{tag,r}} \bmod p$.

$\{L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}_{sk_x}$**:** The values $\{token_{tag,r}\}_{sk_{ID}}$ and $pk_{tag,r}$ have already been generated by the simulator. The value $L_x = \{id = H(x), License\}_{sk_x}$ is a signed license signed by the key sk_x already generated by the simulator. The value $License$ is a license for item, it may contain values such as the period of the license and other limitations. To prevent identity revelation the license must be constructed without using identity information. If this is the case, the simulator can also generate the license.

$\{pk_X, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}_{sk_{TGC}}$**:** The values pk_x, L_x, $\{token_{tag,r}\}_{sk_{ID}}$ and $pk_{tag,r}$ have already been generated by the simulator. The message is then signed

using the signing oracle.

$\{tag, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk_{tag,r}}$**:** The values $sk_{tag,r}$ and tag have already been generated by the simulator. The simulator can construct the value $\{token_{tag,c}\}_{sk_{ID}}$ using a second constructed identity value I_c that is chosen randomly and the signing oracle. The simulator will then construct the one time public key for the customer and tag by choosing a random value $sk_{tag,c}$ and calculating $pk_{tag,c} = g^{-sk_{tag,c}} \bmod p$.

$\{challenge, response\}_{sk_{tag,r}}$**:** The simulator can generate the $challenge$ value using a hash function and the information on the date and time as well as the $token_{tag,r}$ previously created. The $response$ value is generated using the values used to generate $token_{tag,r}$. The message is then signed using the key $sk_{tag,r}$ which has already been generated by the simulator.

5.7.5 Linkability

To show the adversary does not have a chance greater than $\frac{1}{2}$ of guessing the random order, I show that all the messages sent by a reseller r are computationally indistinguishable from the messages sent by another reseller c.

Constant values ($id = H(x)$, L_x, and pk_x) and computationally indistinguishable random values ($pk_{tag,r}$) do not provide linkability information between different runs of the protocol. In any particular run of the protocol, a reseller denoted r sends the following messages (if the customer resells):

$$\{id, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}$$

$$\{\{pk_x, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}_{sk_{TGC}}, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk_{tag,r}}$$

$$\{challenge, response\}_{sk_{tag,r}}$$

The values $id = H(x)$ and L_x are constant for every participant (assuming no identifying information in L_x). The public keys are computationally indistinguishable if the private keys are chosen randomly. I now concentrate on the identity token values $\{token_{tag,r}\}_{sk_{ID}}$, $\{token_{tag,c}\}_{sk_{ID}}$, and $\{challenge, response\}_{sk_{tag,r}}$.

When given an identity token and two different challenge and response values the identity is revealed by design, so I concentrate on the case where only a single challenge response value has been issued for each identity token. Suppose an adversary exists that can break linkability with non negligible probability, then this adversary can be used to discern between computationally indistinguishable

values. The adversary is initially given a signed identity token $\{token_{tag1,r}\}_{sk_{ID}}$. The adversary is then given two other identity tokens $\{token_{tag2,r}\}_{sk_{ID}}$ and $\{token_{tag,c}\}_{sk_{ID}}$ in a random order. The adversary should not have greater chance of guessing the random order than $\frac{1}{2}$.

I now show that all elements in $\{token_{tag1,r}\}_{sk_{ID}}$, $\{token_{tag2,r}\}_{sk_{ID}}$ and $\{token_{tag,c}\}_{sk_{ID}}$ are computationally indistinguishable. The value $\{token_{tag1,r}\}_{sk_{ID}}$ is composed of $A_{tag1,r}$, $B_{tag1,r}$, and $sign\{A_{tag1,r}, B_{tag1,r}\}$.

$A_{tag1,r} = (I_r g_2)^s$ where $I_r = g_1^{u_r}$, g_1 and g_2 are generators of the group and u_r and s are random variables. The value I_r is computationally indistinguishable from the value I_c. The values $A_{tag1,r}$, $A_{tag2,r}$, and $A_{tag,c}$ are composed of Probabilistic Polynomial Time (PPT) operations applied to computationally indistinguishable values and so are computationally indistinguishable.

$B_{tag1,r} = g_1^{x_1} g_2^{x_2}$ where g_1 and g_2 are generators of the group and x_1 and x_2 are random variables. So the values $B_{tag1,r}$, $B_{tag2,r}$, and $B_{tag,c}$ are computationally indistinguishable.

The signed message $sign\{A_{tag1,r}, B_{tag1,r}\}$ contains four values $z_{tag1,r}$, $a_{tag1,r}$, $b_{tag1,r}$, $r_{tag1,r}$. The four values are PPT operations applied to values that are either randomly generated or a generator raised to the power of a random value. So the values $sign\{A_{tag1,r}, B_{tag1,r}\}$, $sign\{A_{tag2,r}, B_{tag2,r}\}$, and $sign\{A_{tag,c}, B_{tag,c}\}$ are computationally indistinguishable.

The only other message sent in the protocol that could provide linkability information is $\{challenge, response\}_{sk_{tag,r}}$. The challenge value is a hash value that can be calculated by any party with access to the identity of the TGC and the current date and time. The response value is a combination of PPT operations applied to random numbers, so the response values of different participants are computationally indistinguishable.

5.8 Modelling

To check the security properties of the tagged transaction protocol, I use the Failures Divergences Refinement (FDR) model checker [33, 78]. The FDR model checker checks a Concurrent Sequential Processes (CSP) [46] model of a protocol against a CSP specification. As discussed in Section 4.6, the adversary has complete control of the network. I also make use of safe simplifying transformations [48]. As the registration step has not changed, this model is

not repeated. The two models I construct examine the properties of spoofing, fabrication, cloning, and network sniffing.

5.8.1 Supplier Generating Tag

In the second phase of the extended tagged transaction protocol, the supplier generates a tag with the TGC. The reseller sends the supplier the one time public key for the tag and its identity token. The supplier then generates a signed license for the item. The supplier sends a signed message with the license, identity token, and the one time public key to the TGC. The TGC then creates and signs the tag and sends it back to the supplier who sends it to the reseller. The high level description of the protocol is:

$Reseller \rightarrow Supplier : item, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}$
$Supplier \rightarrow TGC : \{L_x = \{H(x), License_x\}_{sk_x}, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}_{sk_x}$
$TGC \rightarrow Supplier : \{pk_x, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}_{sk_{TGC}}$
$Supplier \rightarrow Reseller : \{pk_x, L_x, \{token_{tag,r}\}_{sk_{ID}}, pk_{tag,r}\}_{sk_{TGC}}$

I apply the following safe simplifying transformations before modelling the supplier generating a tag with the TGC:

1. Removal of the encryption on the field $L_x = \{H(x), License_x\}_{sk_x}$ to make it $\{H(x), License\}$.
2. Removal of the atomic field $License$.
3. Removal of the atomic field $H(x)$.
4. Removal of the atomic field $item$.
5. Removal of the encryption on the field $\{token_{tag,r}\}_{sk_{ID}}$ to make it $token_{tag,r}$.

After the simplification the description of the protocol is:

$Reseller \rightarrow Supplier : token_{tag,r}, pk_{tag,r}$
$Supplier \rightarrow TGC : \{token_{tag,r}, pk_{tag,r}\}_{sk_x}$
$TGC \rightarrow Supplier : \{pk_x, token_{tag,r}, pk_{tag,r}\}_{sk_{TGC}}$
$Supplier \rightarrow Reseller : \{pk_x, token_{tag,r}, pk_{tag,r}\}_{sk_{TGC}}$

In this model, the set $Tags = \{pktagreseller, pktagwrong\}$ represents the one time keys for the tags . As these keys are never used for encryption or signing in this part of the protocol they do not need to be represented as public and private key pairs. Each participant in the protocol has a public and private key pair. The value pk_x represents the public key for the item and sk_x is represents private key. The set $Token = \{tokenreseller\}$ represents the identity tokens. The initial knowledge of the adversary is $pktagwrong$, $tokenreseller$, pk_x, pk_{TGC}, $pk_{Supplier}$, $pk_{Reseller}$, and $sk_{Reseller}$. The model of the protocol for CSP is:

$Reseller \rightarrow Supplier : tokenreseller, pktagreseller$
$Supplier \rightarrow TGC : \{tokenreseller, pktagreseller\}_{sk_x}$
$TGC \rightarrow Supplier : \{pk_x, tokenreseller, pktagreseller\}_{sk_{TGC}}$
$Supplier \rightarrow Reseller : \{pk_x, tokenreseller, pktagreseller\}_{sk_{TGC}}$

The generation of a tag has to resist a fabrication attack where an adversary manages to generate a valid tag. The specification states that if the reseller receives a tag that it accepts, the supplier must have sent a message to the TGC requesting the tag. The FDR model checker returns TRUE after 111 states with 387 transitions.

5.8.2 Reseller Generating Tag

The CSP description for the reseller generating a tag in the extended tagged transaction protocol is now examined. The customer sends the reseller the one time public key for the tag and its identity token. The reseller then sends the tag from the supplier, the customer's one time public key and the customer's identity token all signed by the one time private key of the tag to the TGC. The TGC then sends a challenge to the reseller. The reseller returns a signed message with the challenge and response values. The TGC creates and signs the tag and sends it back to the reseller who sends it to the customer. The high level description of the protocol is:

$Customer \rightarrow Reseller : item, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}$
$Reseller \rightarrow TGC : \{\{pk_x, L_x, pk_{tag,r}\}_{sk_{TGC}}, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk_{tag,r}}$
$TGC \rightarrow Reseller : challenge$
$Reseller \rightarrow TGC : \{challenge, response\}_{sk_{tag,r}}$

$TGC \rightarrow Reseller : \{pk_x, L_x, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk_{TGC}}$
$Reseller \rightarrow Customer : \{pk_x, L_x, \{token_{tag,c}\}_{sk_{ID}}, pk_{tag,c}\}_{sk_{TGC}}$

Before we model the reseller generating a tag with the TGC we apply the following safe simplifying transformations:

1. Removal of the encryption on the field $L_x = \{H(x), License_x\}_{sk_x}$ to make it $\{H(x), License\}$.
2. Removal of the atomic field $License$.
3. Removal of the atomic field $H(x)$.
4. Removal of the atomic field $item$.
5. Removal of the encryption on the field $\{token_{tag,c}\}_{sk_{ID}}$ to make it $token_{tag,c}$.
6. Removal of the encryption on the field $\{token_{tag,r}\}_{sk_{ID}}$ to make it $token_{tag,r}$.
7. Redirecting a message that is sent via a third party to go direct. We direct message 5 to go straight from the TGC to the customer and not via the reseller.

After the simplification the description of the protocol is:

$Customer \rightarrow Reseller : token_{tag,c}, pk_{tag,c}$
$Reseller \rightarrow TGC : \{\{pk_x, token_{tag,r}, pk_{tag,r}\}_{sk_{TGC}}, token_{tag,c}, pk_{tag,c}\}_{sk_{tag,r}}$
$TGC \rightarrow Reseller : challenge$
$Reseller \rightarrow TGC : \{challenge, response\}_{sk_{tag,r}}$
$TGC \rightarrow Customer : \{pk_x, token_{tag,c}, pk_{tag,c}\}_{sk_{TGC}}$

I model the earlier generation of a tag ($\{pk_x, token_{tag,r}, pk_{tag,r}\}_{sk_{TGC}}$) by the introduction of an extra set $ActualTags = \{tag, tag2, tagwrong, tagwrong2\}$ that represents the tags that have been sent from the supplier to the reseller. To represent the reseller signing the tag with the one time key embedded in the tag, the reseller signs the message with its own private key. To include the complete generation of a tag in this model makes the size of the model intractable. The set $Tags = \{pktag, pktag2, pktagwrong, pktagwrong2\}$ represents the one time keys for the new tags. The set $Token = \{tktag, tktag2\}$ represent identity tokens. The set $Challenge = \{c_1, c_2\}$ represents the challenge values and the

set $Response = \{r_1, r_2\}$ represents the response values. The item key pk_x is represented as a public key with private key sk_x. To model the possibility of a cloning attack, two runs of the protocol happen at the same time. The initial knowledge of the adversary is $tagwrong$, $tagwrong2$, $pktagwrong$, $pktagwrong2$, $tktag$, $tktag2$, c_1, c_2, r_1, r_2, pk_x, pk_{TGC}, $pk_{Supplier}$, and $pk_{Reseller}$. The model of the protocol for CSP is:

$Customer \rightarrow Reseller : tktag, tktag2, pktag, pktag2$
$Reseller \rightarrow TGC : \{tag, tktag, pktag, tag2, tktag2, pktag2\}_{sk_R}$
$TGC \rightarrow Reseller : c_1, c_2$
$Reseller \rightarrow TGC : \{c_1, r_1, c_2, r_2\}_{sk_R}$
$TGC \rightarrow Customer : \{pk_x, tktag, pktag\}_{sk_{TGC}}, \{pk_x, tktag2, pktag2\}_{sk_{TGC}}$

There are two specifications for the reseller generating the tag. The first specification prevents cloning where an adversary creates two new tags from the same initial tag. The specification states that if the customer receives two tags, then two different values must have been signed by the TGC. The FDR model checker returns TRUE after 457 states with 2005 transitions. The second specification checks for a network sniffing attack where an adversary sees a tag being sent over the network and tries to use it to generate a new tag. The specification states that if a customer receives tags signed by the TGC then the reseller must have sent a message to the TGC to request the generation of the tags. The FDR model checker returns TRUE after 1423 states with 4835 transitions.

5.9 Summary

This chapter has presented the extended tagged transaction protocol. The extended tagged transaction protocol uses ideas from digital cash where double spenders can be identified. In the extended tagged transaction protocol, if a reseller tries to clone a tag their identity is revealed. Once the identity of a malicious reseller is known, they can be prevented from generating more identity tokens.

The extended tagged transaction protocol uses the TGC as both the bank and the merchant in the digital cash model. A reseller or customer will need to obtain a restricted blind signed identity token in a protocol with the TGC before they

can purchase any items. This signed identity token is then added to the tags that are generated and used by the extended tagged transaction protocol.

A thorough security analysis has been carried out showing that the extended tagged transaction protocol prevents spoofing, fabrication, network sniffing, cloning, identity revelation, and linkability attacks. Finally, the details and results of the modelling of the extended tagged transaction protocol, using the FDR model checker, show the protocol prevents spoofing, fabrication, cloning, and network sniffing attacks.

The security analysis in the past two chapters assumes an anonymous communication channel and that the TGC is acting correctly. In the next chapter, I remove the assumption that the TGC is acting correctly and discuss ways to provide an anonymous communication channel and verify the actions of the TGC.

Chapter 6

Anonymity and Tag Generation Centre Distribution and Verification

This chapter discusses methods to provide anonymous communication and the distribution and verification of the Tag Generation Centre (TGC). When resellers (and suppliers when they are anonymous) are communicating with the TGC they use an anonymous communication channel. This prevents the TGC from building up large amounts of data on the transactions of a reseller.

In previous chapters, the TGC has been assumed to be acting as a single trusted third party. In this chapter, I remove this assumption and present different methods for distributing the TGC over multiple parties and verifying the actions of the TGC. Having the TGC distributed over multiple parties improves reliability by not providing a single point of failure. Verifying the actions of the TGC means that the TGC is no longer acting as a trusted third party. Properties of these different methods are discussed, and in the next chapter the performance of the methods are examined.

6.1 Anonymity

Research on anonymous communications began with a paper by David Chaum in 1981 [21]. Since then there has been a large body of research both developing anonymous communication protocols and analysing and attacking these protocols. Anonymity has been defined in work by Pfitzmann et al as *the state of not being identifiable within a set of subjects, the anonymity set* [72]. In the tagged transaction protocol, the set of subjects includes resellers and anonymous

suppliers.

For the tagged transaction protocol, I present three different methods of providing an anonymous communication channel:

1. Anonymity is provided by the TGC which hides all identification information about anonymous parties.

2. Anonymity is provided by a second party. This two party style of anonymity prevents the TGC from learning the identities of anonymous parties.

3. Anonymity is provided by an anonymous communication channel.

6.1.1 Anonymity provided by the TGC

Anonymity can be provided by the TGC. When the TGC provides anonymity, anonymous parties in the tagged transaction protocol communicate directly with the TGC. The TGC then knows the identity of all resellers and suppliers in the protocol. When the TGC publishes or sends tags and other information, it does not publish the identifying information of the participant that sent the message. The messages sent as part of the tagged transaction protocol and the extended tagged transaction protocol do not contain identifying information as shown in Section 4.5.5 and Section 5.7.4.

While the anonymity provided by the TGC prevents any external observers from learning the identities of the participants in the protocol, it does not prevent the TGC from learning the identities. The TGC can then build up detailed records of the transactions of each reseller and supplier in the tagged transaction protocol. This may be commercially sensitive information and the TGC must be trusted to not reveal it.

An advantage of the TGC providing anonymity is that any resellers that are acting maliciously or incorrectly can be identified.

6.1.2 Two Party Anonymity

Two party anonymity uses a second party, an anonymity service, to provide anonymity in communication between the resellers and anonymous suppliers and the TGC. Figure 6.1 shows the process for sending messages between a reseller and the TGC using two party anonymity. The reseller sends the message

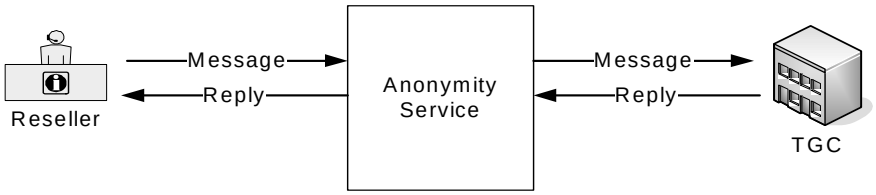

Figure 6.1: Two Party Anonymity

to the anonymity service which sends it on to the TGC. The TGC then replies to the anonymity service which forwards the reply on to the reseller. As all messages to the TGC come from the anonymity service, the TGC can gain no information on the reseller.

Some systems already provide such an anonymity service. An early example is the anon.penet.fi relay (no longer running). Later examples that are still running include Anonymizer[1], KProxy[2], and Hide My Ass![3] among many others.

Similar to anonymity provided by the TGC, the use of two party anonymity allows the anonymity service to build up records of the participants in the tagged transaction protocol. The use of the two party anonymity prevents external parties from building up records of the actions of the participants in the protocol. Depending on the terms and conditions of the anonymity service, a malicious reseller can be identified.

6.1.3 Anonymous Communication Channel

Anonymous communication channels have been developed to prevent any party being able to trace the identity of the senders of messages. These systems use a group of servers or relays. No single server has the information to link the sender of a message with the message that is received by the receiver. As long as one of the relays in the chain is honest, it is hard to link the message and its sender.

Mix networks were first suggested by Chaum [21]. A mix network involves a party sending an item to a mix node (or a series or cascade of mix nodes) who then sends it to the addressee. The purpose of the mix node is to hide the correspondences between the items in its input and those in its output. Figure 6.2 shows a cascade of mix nodes. Suppose we have a party A that wants to send a

[1]http://www.anonymizer.com/
[2]http://kproxy.com/
[3]http://hidemyass.com/

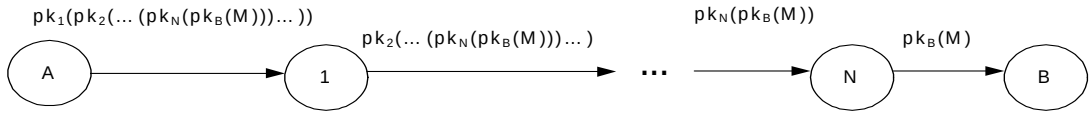

Figure 6.2: A Cascade of Mix Nodes

message M to the recipient B anonymously through a series of $1, ..., N$ mix nodes. A will form a message M that is encrypted by the public key of B. Then A will add an encryption layer for each mix node $1, ..., N$ using their public keys. As each mix node receives the message it will wait to receive a batch of messages before re-arranging, or mixing, the order of the messages, stripping off one of the encryption layers, and forwarding it on to the next mix node. Finally, B will receive a message encrypted with its public key.

Onion routing [88] differs from the standard mix network by establishing a stream connection between the sender and recipient. Messages can be sent to open or close a circuit through the network. Once a circuit is established, messages with the same label are sent through the same circuit. The initial message through the network is encrypted in layers and can only be decrypted by the nodes chosen by the sender. Once the circuit has been established data travelling through the network is encrypted with symmetric keys. The mixing strategy used by each of the nodes is very close to a first in first out mixing. This means that onion routing is susceptible to a traffic analysis attack, which is made easier in the absence of large amounts of traffic. This minimal mixing strategy was employed to provide maximum real time performance.

TOR [28] is a second generation onion router. TOR is also designed to provide real time anonymous web browsing. Instead of the layered onion approach used by the original onion network to establish a circuit through the network, TOR uses an iterative mechanism. A sender in the TOR network connects to the first node in the circuit it wants to establish, then requests that node connects to the next one. An authenticated Diffie-Hellman key exchange is used to establish symmetric keys for the stream. The TOR network does not claim to provide anonymity in the presence of passive global adversaries but TOR does provide low latency and there are currently over 2000 TOR publicly available TOR nodes [4].

The type of anonymous communication channel provided by mix networks

[4]https://metrics.torproject.org/network.html

and the TOR network prevent any party from being able to build up records of the identities of the senders of messages sent to the TGC. This is in contrast to the TGC providing anonymity or two party anonymity where the information to link the sender with a message is held by the TGC or anonymity service. A mix network has no way of revealing the identity of a reseller if malicious actions are detected by the TGC.

6.2 TGC Distribution and Verification

The choice of the distribution of the TGC affects the verification mechanism used to verify the actions of the TGC. I present two main distributions and methods of verification for the TGC.

1. Single party TGC. The TGC is a single well known party or a group of well known parties where only one is chosen as the TGC for a certain supplier. Verification of the TGC uses a public bulletin board.

2. Multiple party TGC. The TGC is a group of independent servers. Verification is provided by threshold trust, as long as a threshold number of the servers are acting correctly the TGC can be trusted to be acting correctly.

6.2.1 Single Party TGC

A single party TGC is the simplest distribution. All requests from resellers and suppliers for tag generation use this single party TGC. Only one public key is needed to check tags used as part of the tagged transaction protocol. Although a single party is in charge of the TGC, it can be implemented as a distributed service similar to services such as gmail[5] or hotmail[6].

A single party TGC without verification would be a trusted third party. To prevent this, I introduce a verification technique for the TGC. The TGC has access to a public bulletin board. Only the TGC can write to the board but any party can read what is written on the board. The TGC then posts all messages it sends or receives to the bulletin board. When a reseller or customer receives a tag, they can check the bulletin board to make sure that the tag was originally produced by the supplier and that the tag has not been cloned. While this provides a method

[5]gmail.com
[6]hotmail.com

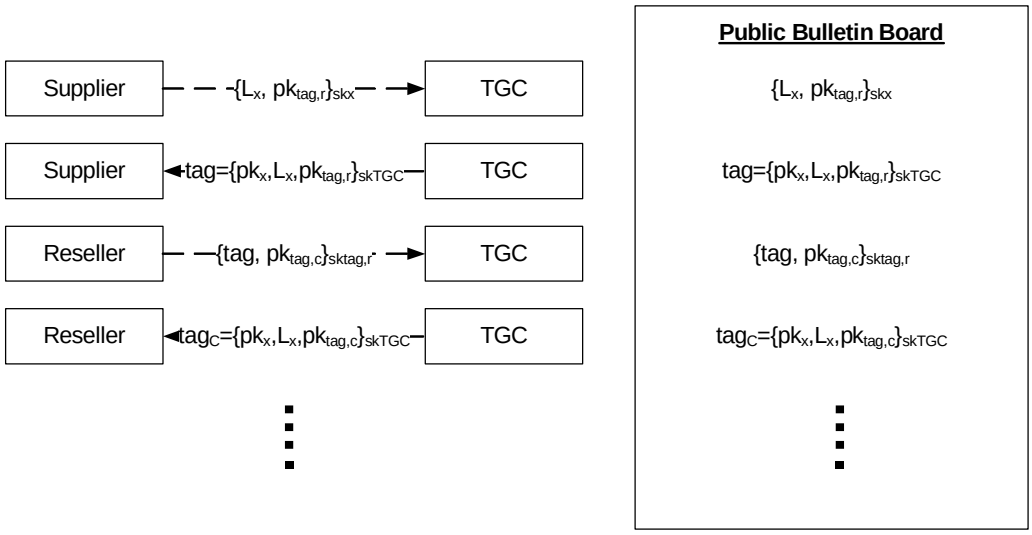

Figure 6.3: Single TGC Verification

to verify the TGC, it leaks the number of resellers between the customer and supplier. It also reveals the terms of the license produced by the supplier.

Figure 6.3 shows the verification process with the TGC publishing all the messages it sends or receives to the bulletin board. A verifier will want to check that the tag they have received was first created by the supplier and that it has not been cloned. The tag the customer receives will be part of a chain of tags denoted $\{tag_{customer}, tag_{reseller1}, ..., tag_{resellern}\}$ where there are n resellers in the chain. To verify the actions of the TGC, a customer will need to check that, for each tag in the chain, the old tag has only been used to generate one new tag. The customer will also check that the initial tag $tag_{resellern}$ was generated by a request from the supplier. Assuming the chance of the TGC cheating is $\frac{1}{n}$, and the chance of a customer verifying the TGC is $\frac{1}{m}$, then the chance of a cheating TGC being caught is $\frac{1}{mn}$. If the TGC cheats every time and every customer verifies the TGC, then the chance of a cheating TGC being caught is 1. If the TGC cheats once every thousand times, and one of every thousand customers verifies the TGC, then the chance of a cheating TGC being caught is $\frac{1}{1000000}$. The TGC has more motivation to act maliciously for high value items, so customers should verify the actions of the TGC more often for high value items to detect cheating with higher probability.

A second option for a single party TGC is to use multiple independent TGCs. When a supplier first registers an item, it will choose a TGC to use for this item. The supplier then advertises this choice using some out of band method

such as its website. When a customer purchases an item from a supplier, they check on the supplier's web site to see what TGC they should use. This allows suppliers to use a TGC that they trust. For example, if Google was in charge of the TGC, Microsoft may not want to use it for their products. The use of multiple independent TGCs also spreads the load over the group. To use multiple independent TGCs, the suppliers must be known: having multiple independent TGCs does not support supplier anonymity. The supplier could provide their own TGC for their items. If the supplier provides their own TGC, then the supplier must be online for transactions. This removes the offline supplier property of the tagged transactions protocol.

6.2.2 Multiple Party TGC

Rather than having the TGC as a single party, the TGC can be distributed as a group of independent heterogeneous servers. A customer will only accept a tag if it has been signed by a threshold value of the multiple TGCs. For example, the customer could accept a tag if it is signed by three out of five TGCs. This would improve availability by only requiring a threshold number of the TGCs to be online to generate a license. To sign a tag using a threshold value of TGCs, either each TGC has a separate key that they use to sign a tag, or the TGCs use a threshold group signature scheme.

Group signature schemes were first introduced by Desmedt et al [27]. In a threshold group signature scheme, a group of n signers each have access to a part of a shared secret. To sign a message requires the co operation of t out of n signers. It is known that t of the n signers were involved in generating the signature but not which of the t out of n signers. The development of threshold group signatures began with schemes that need a trusted centre [27] but later schemes have removed this requirement [47, 54, 58, 70]. Some threshold group signature schemes support traceability of a signer [54], this property could be used to detect what TGC has signed an invalid tag. Threshold group signature schemes involve broadcasting messages between the group of signers.

I use individual keys and signatures instead of threshold group signatures for two reasons: it allows an individual TGC that is acting incorrectly to be easily identified, and it requires less co-ordination between the multiple independent TGCs to sign a tag. The use of individual signatures will result in more network traffic between the TGCs and the reseller as each TGC will send an independent

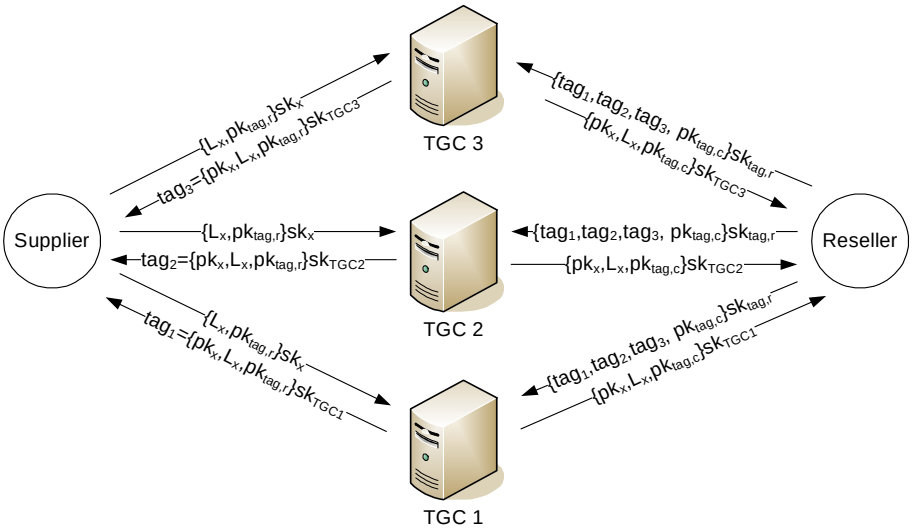

Figure 6.4: Supplier and Reseller Interacting with Multiple TGCs

signature over the network rather than having a single signature from the group. When a supplier or reseller wants to generate a tag, they will have to send the tag request message to at least the threshold number of TGCs. The TGCs will then each independently reply as shown in Figure 6.4. When a reseller is generating a tag, the reseller will need to include all tags signed by the group of TGCs (there must be at least the threshold number of them).

In the multiple TGC distribution, the actions of an individual TGC are not verified. As long as fewer than the threshold value of TGCs are acting maliciously, suppliers, resellers, and customers can have confidence that the TGCs are acting correctly.

When using a group of TGCs, it is possible that the information seen over the network may be different for each TGC. This could be caused by a network outage for one of the TGCs, or malicious actions such as a denial of service attack. This would result in the state of the TGCs being different, so it is possible that one of the TGCs has not seen a reseller generating a tag and still has the old tag in its database. If the TGC is acting correctly and it receives a tag generation request using a new tag that it does not have in its database, it will not generate the new tag.

If a TGC receives a tag request for a tag that is not in its database, it queries the other TGCs for the tag chain for the tag. The other TGCs will send all the messages they have received from suppliers and resellers for this tag. The other

TGCs do not need to send a record of what they have sent as these messages can be recreated from the messages received from the suppliers and resellers. Each of the TGCs sign the messages with their own private key. The TGC that has been offline must receive the records of messages from at least the threshold number of other TGCs to prevent malicious TGCs from being able to give it false information. Using this method, a TGC in the group that goes offline for a period of time can synchronise its view with the other TGCs.

A standard (t, n) threshold scheme has $n = 2t - 1$ as the threshold value. This means that as long as the majority of participants are honest, the user can trust the result. In the tagged transaction protocol, an adversary can clone a tag when using multiple TGCs as shown in Figure 6.5 when using the standard threshold value. Suppose a group of three TGCs (TGC 1, TGC 2, and TGC 3) and a threshold value of two (a $(2, 3)$ threshold scheme) are being used. One of these TGCs (TGC 2) is malicious and is acting together with an adversary trying to clone a tag. Suppose the adversary already has tags:

$$tag_1 = \{p_x, L_x, pk_{tag,r}\}_{sk_{TGC1}}$$

$$tag_2 = \{p_x, L_x, pk_{tag,r}\}_{sk_{TGC2}}$$

$$tag_3 = \{p_x, L_x, pk_{tag,r}\}_{sk_{TGC3}}$$

The adversary then wants to use the malicious TGC to clone two new tags with the values $pk_{tag,c1}$ and $pk_{tag,c2}$. The adversary sends the following message to TGC 3 and the malicious TGC 2:

$$\{tag_1, tag_2, tag_3, pk_{tag,c1}\}_{sk_{tag,r}}$$

As TGC 3 is acting correctly it will generate and return the message:

$$\{pk_x, L_x, pk_{tag,c1}\}_{sk_{TGC3}}$$

The malicious TGC will return:

$$\{pk_x, L_x, pk_{tag,c1}\}_{sk_{TGC2}}$$

The adversary sends the following message to TGC 1 and the malicious TGC 2:

$$\{tag_1, tag_2, tag_3, pk_{tag,c2}\}_{sk_{tag,r}}$$

As TGC 1 is acting correctly it will generate and return the message:

$$\{pk_x, L_x, pk_{tag,c2}\}_{sk_{TGC1}}$$

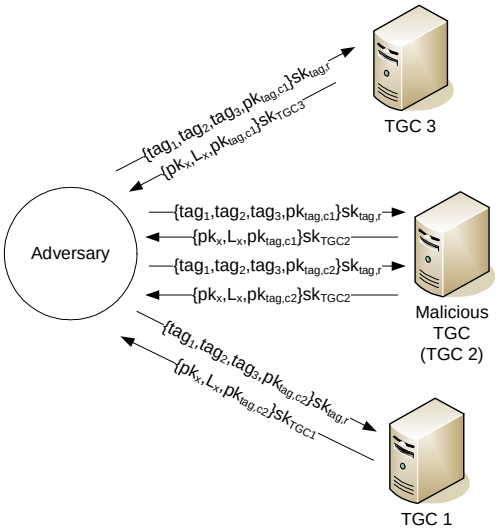

Figure 6.5: Adversary Cloning Tag with Multiple TGCs

The malicious TGC will clone the tag and generate the message:

$$\{pk_x, L_x, pk_{tag,c2}\}_{sk_{TGC2}}$$

The adversary then has cloned two tags from an initial tag using a single malicious TGC. This has broken the tagged transaction protocol by allowing a cloning attack when only one out of three TGCs was malicious.

To prevent the attack shown in Figure 6.5, I alter the threshold value that is used. The adversary (or adversaries) are able to split the honest parties in to two partitions and give both different information. To prevent this, the threshold must be set so that all the malicious parties and half the honest parties are not enough to sign tags. The formula for calculating the number of participants n when given a threshold t is $n < 2t - \lfloor \frac{t}{2} \rfloor$. Examples of safe threshold values to use include $(4, 5)$, $(5, 7)$, $(7, 9)$, and $(9, 13)$.

To use multiple independent heterogeneous TGCs, the TGCs have to be pre-selected and well known. The multiple party TGC distribution I have shown is not dynamic. It does not require all the TGCs to be online at the same time, but the set of TGCs remains static. The parties in charge of the TGCs must be carefully chosen. While the threshold model prevents a single malicious or faulty TGC from breaking the protocol, it is best to choose parties to run the TGCs that are trustworthy. The choice of TGCs may also reflect commercial interests where the TGCs may be run by a group of the biggest companies in a particular

industry. Other issues include ensuring diversity among the multiple TGCs to prevent them being sensitive to the same attack.

Introducing multiple TGCs will increase the computational and communication complexity of both the tagged transaction protocol and the extended tagged transaction protocol. The growth in the complexity of the tagged transaction protocol when using multiple TGCs is caused by:

- When the reseller sends a tag generation request to the TGCs it must send the request to at least the threshold number of TGCs and it must include at least a threshold number of signed tags with the request.

- When a TGC receives a tag generation request from a reseller it needs to check at least a threshold number of signed tags before it will generate the new tag.

The design decision to use multiple TGCs with individual signing keys will result in the complexity of both the tagged transaction protocol and the extended tagged transaction protocol being $O(t^2)$. The use of a threshold group signature scheme will only require a single tag be sent to the group of TGCs but will increase the complexity of the TGCs co-operating to sign a tag. More details on the complexity and performance impacts of increasing the number of TGCs can be found in chapter 7.

The group of TGCs will be constant and chosen before the protocol starts. The purpose of multiple TGCs is to distribute the trust over multiple independent parties. If an increase in the number of customers or transactions creates a large load on the TGCs and reduces performance then the TGCs would need to be replicated. It is important to differentiate the replication of a TGC from an increase in the number of TGCs in the group. Replicating a TGC will involve using a technique such as a server cluster to increase the performance of the TGC but will not result in an increase in the number of TGCs in the group. The cost of adding replicas is maintaining consistency between replicas of the TGC so that each replica would return the same answer to a query.

Another method of handling scalability issues is to separate the marketplace in to separate sub-marketplaces. For example, one marketplace could deal with music, one with video, and one with eBooks. One group of multiple TGCs could then be in charge of each sub-marketplace. This would split the load of the whole marketplace over several groups of multiple TGCs in charge of sub-marketplaces.

6.3 Summary

In this chapter, I have presented options for the anonymous communication channel and the distribution and verification of the TGC. The choices for both the anonymity and the distribution provide different characteristics in terms of computational and communication complexity, verification and anonymity properties, and legal aspects of anonymity.

I describe three options for the anonymous communication channel: the use of the TGC as an anonymity server, the use of a third party anonymity server, and the use of an anonymous communication channel based on mix networks. The use of the TGC or a third party as the anonymity server allows revocable anonymity for suppliers and resellers in the tagged transaction protocol. Depending on legal requirements, this may be a necessary property. Anonymous communication using a mix network, such as the TOR network, provides the strongest anonymity properties. However, it is not possible to provide perfect anonymity with any anonymous communication channel. When viewing the anonymity network as a black box that provides perfect anonymity, if an adversary can watch both ends of the anonymous communication, persistent communication between two parties will be detected. I use both the TGC and the TOR network to provide anonymity for the tagged transaction protocol. The TOR network has a large number of active nodes that can be used for testing. I compare the performance of the tagged transaction protocol using both of these approaches in the next chapter.

The distribution of the TGC affects the properties and mechanism of the verification of the TGC. A contribution of this work is to provide methods to prevent the TGC acting as a trusted third party. I have presented two main options for verification. The first option is a single (or many single independent) TGC that uses a public bulletin board to provide verification. This technique leaks the number of links between the supplier and the customer. To verify the actions of the TGC the reseller will have a higher communication and computational complexity as it has to download and verify the signatures of the messages on the board. The second option is to use multiple TGCs with a threshold value of them assumed to be acting correctly. As long as this threshold value of TGCs are acting correctly, the actions of the TGC are verified. The multiple party TGC distribution does not leak the number of links between the supplier and customer or require a public bulletin board and customer verification of data. However,

the multiple party TGC distribution has a greater communication complexity between the TGCs to prevent cloning attacks and provide methods for an offline TGC to synchronise with the other TGCs. The multiple party TGC distribution also provides fault tolerance as a certain percentage of the TGCs (depending on the threshold value) can fail and suppliers and resellers can still generate tags. In the following chapter, I present complexity and performance results for the two distribution options.

Chapter 7

Complexity and Performance

This chapter presents complexity and performance results for both the tagged transaction protocol and the extended tagged transaction protocol. These results show the relationship between the distribution of the TGC and the anonymous communication channel presented in the previous chapter and the performance of the protocol. This chapter initially shows the computational and communication complexity. These results are used to compare the tagged transaction protocol and the extended tagged transaction protocol to anonymous credentials, a protocol that can anonymously provide provenance in reseller chains.

The second part of this chapter details the implementation of the tagged transaction protocol and the extended tagged transaction protocol. The details of the experimental setup are described and the results of the performance tests done on both the tagged transaction protocol and the extended tagged transaction protocol are presented. The performance tests shows the effects of changing the key size of the signature scheme, the distribution of the TGC, the number of resellers in the chain, and the use of the TOR anonymous communication channel on the time taken to complete the protocol.

7.1 Complexity

In this section I present the computational and communication complexity of the tagged transaction protocol. This does not include any extra communication or computation that is necessary to provide an anonymous communication channel. Table 7.1 shows the computational complexity for the tagged transaction protocol and Table 7.2 shows the computational complexity for the extended tagged

transaction protocol. These tables use the following notation:

- p and q are large primes where $p = 2q + 1$. The operations for the digital signature scheme are done in the group $\mathbb{G}_p$.
- n is the size of the modulus used for the encryption scheme.
- x is the complexity of modular exponentiation.
- y is the complexity of modular division.
- t is the threshold number of TGCs used.
- T is the total number of TGCs used.
- r is the number of resellers in the chain.

The complexity results make the following assumptions: the size of the modulus for the encryption and group for the signatures are equal ($|n| = |p|$), and the registration uses the worse case ($|p| = |n| = 1024$) value.

	Customer	Reseller	Supplier	TGC	Total
Registering Item			$y + 3xT$	$3xT$	$y + 6xT$
Verifying Tag	$4xt$				$4xt$
Generating Tag		$y + 4xt$	$6xt$	$5xt$	$y + 15xt$
Reseller Generating Tag	$y + 4xt$	$ry + 9xrt$		$4xt^2 + xrt$	$y + 4xt + 6xrt + 4xt^2$
Total (no registration)	$y + 4xt$	$ry + 9xrt$	$6xt$	$5xt + xrt + 4xt^2$	$y + ry + 15xt + 10xrt + 4xt^2$

Table 7.1: Tagged Transaction Protocol Computational Complexity

There is a linear relationship between the computational complexity of registering an item and the total number of Tag Generation Centres (TGCs). This is because the item needs to be registered with every TGC.

To verify a tag requires verifying two separate digital signatures on each tag. This will require four modular exponentiations for each tag returned by one of the threshold number of TGCs.

The relationship between the computational complexity of generating a tag and the threshold number of TGCs is linear. A signed tag request will have to be sent from the supplier to the threshold number of TGCs who will sign and return the tag.

To regenerate a tag, all the resellers in the chain have to verify a signature on a tag for the threshold number of TGCs and sign a tag regeneration request for the

threshold number of TGCs, which would lead to a linear relationship between the computational complexity and the threshold number of TGCs. However, each TGC has to check the signature on a threshold number of signed tags for each reseller. This means that there is a linear relationship between the computational complexity and the square of the threshold number of TGCs.

The final total value shows the computational complexity of the entire reseller chain, not including the registration of the item. The registration step is not included as this only needs to be done once by the supplier and not for every transaction. There is a linear relationship between the square of the threshold number of TGCs and the computational complexity of the tagged transaction protocol and a linear relationship between the number of resellers in the chain and the computational complexity.

	Customer	Reseller	Supplier	TGC	Total
Verifying Tag	$8xt$				$8xt$
Registering Account	x			x	$2x$
Generating ID Token	$y + 12x$			$2x$	$y + 14x$
Checking ID Token	$4x$				$4x$
Generating Tag		$y + yt + 18xt$	$10xt$	$9xt$	$y + yt + 37xt$
Reseller Generating Tag	$y + yt + 16xt$	$ry + ryt + 35xrt$		$8xt^2 + 6xrt$	$y + 2yt + 28xt +$ $11xrt + 8xt^2$
Total (no registration)	$y + yt + 24xt$	$ry + ryt + 35xrt$	$10xt$	$9xt + 6xrt + 8xt^2$	$y + yt + ry + ryt +$ $41xrt + 43xt + 8xt^2$

Table 7.2: Extended Tagged Transaction Protocol Computational Complexity

To verify a tag requires verifying two separate digital signatures on each tag and the signed identity token. This will require eight modular exponentiations for each tag returned by one of the threshold number of TGCs.

Registering an account only requires one modular exponentiation by the customer (or reseller) registering the account and the TGC. Generating an ID Token requires twelve modular exponentiation and one modular division from the customer (or reseller) and two modular exponentiation from the TGC. Checking an ID Token requires four modular exponentiations.

Generating a tag has a linear relationship between the computational complexity and the threshold number of TGCs. This is because the tag generation request has to be sent to the threshold number of TGCs from the supplier.

To regenerate a tag requires each TGC to check the signatures for a tag for a threshold number of tags before they will generate a new tag. This causes a

linear relationship between the computational complexity and the square of the threshold number of TGCs.

There is a linear relationship between the computational complexity of the extended tagged transaction protocol and the number of resellers. There is a linear relationship between the computational complexity of the extended tagged transaction protocol and the square of the threshold number of TGCs. The extra computations compared to the tagged transaction protocol are required as each customer and reseller must generate a signed identity token with the TGCs before getting signed tags. This increases the number of modular exponentiations and modular divisions that need to be done to generate a tag.

Table 7.3 and Table 7.4 show the communication complexity for the tagged transaction protocol and the extended tagged transaction protocol. In this analysis the communication complexity refers to the amount of data that is sent by each participant in the protocol. The following notation is used to show the communication complexity:

- p and q are large primes where $p = 2q + 1$. The operations for the digital signature scheme are done in the group $\mathbb{G}_p$.
- n is the size of the modulus used for the encryption scheme.
- h is the size of the output of the hash function used to identify items (or the size of the output of an alternate identification scheme).
- l is the size of the license for the item.
- t is the threshold number of TGCs used.
- T is the total number of TGCs used.
- r is the number of resellers in the chain.

	Customer	Reseller	Supplier	TGC	Total
Size of Tag					$h + l + 2p + 4q$
Registering Item			$3Tn$	$h + p + 2q$	$3Tn + h + p + 2q$
Generating Tag		$h + p$	$t(h + l + p + 4q)$	$t(h + l + 2p + 4q)$	$h + p + t(2h + 2l + 3p + 8q)$
Reseller Generating Tag	$h + p$	$t(p + 2q)+$ $t^2(h + l + 2p + 5q)$		$t(h + l + 2p + 4q)$	$h + p + t(h + l + 3p + 6q)+$ $t^2(h + l + 2p + 5q)$
Total (Without Registration)	$h + p$	$h + p + t(p + 2q)+$ $t^2(h + l + 2p + 5q)$	$t(h + l + 2p + 4q)$	$2t(h + l + 2p + 4q)$	$2h + 2p+$ $t(3h + 3l + 7p + 14q)+$ $t^2(h + l + 2p + 5q)$

Table 7.3: Tagged Transaction Protocol Communication Complexity

Table 7.3 shows the communication complexity of the tagged transaction protocol. There is a linear relationship between the total number of TGCs and the communication complexity to register a tag. This is because the registration message needs to be sent to every TGC in the group. There is a linear relationship between the communication complexity of generating a tag and the threshold number of TGCs as the supplier will need to send the generation request to the threshold number of TGCs and return the threshold number of signed tags to the reseller. The total communication complexity has a linear relationship with the square of the threshold number of TGCs. This is because each TGC needs to be sent a threshold number of tags before it will sign a new tag.

	Customer	Reseller	Supplier	TGC	Total
Size of Tag	$h + l + 7p + 5q$				$h + l + 7p + 5q$
Registering Account	tp			tp	$2tp$
Generating ID Token	tq			$t(2p + q)$	$t(2p + 2q)$
Generating Tag		$h + 6p + q$	$t(h + l + 6p + 5q)$	$t(h + l + 7p + 5q)$	$h + 6p + q +$ $t(2h + 2l + 12p + 10q)$
Reseller Generating Tag	$h + 6p + q$	$t(5p + 3q) +$ $t^2(h + l + 7p + 5q)$		$t(h + l + 7p + 5q)$	$h + 6p + q +$ $t(h + l + 12p + 8q) +$ $t^2(h + l + 7p + 5q)$
Total (Without Registration)	$h + 6p + q$	$h + 6p + q +$ $t(5p + 3q) +$ $t^2(h + l + 7p + 5q)$	$t(h + l + 6p + 5q)$	$2t(h + l + 7p + 5q)$	$2h + 12p + 2q +$ $t(3h + 3l + 25p + 18q) +$ $t^2(h + l + 7p + 5q)$

Table 7.4: Extended Tagged Transaction Protocol Communication Complexity

Table 7.4 shows the communication complexity of the extended tagged transaction protocol. The communication complexity of registering an account and generating an ID Token has a linear relationship with the threshold number of TGCs because the customer or reseller will need to generate a threshold number of blind signed identity tokens to request a tag. Similar to the tagged transaction protocol, the communication complexity of the total protocol without registration scales linearly with the square of the threshold number of TGCs. This is because the TGCs must be sent a threshold number of tags before they will sign a new one. The extra communication complexity is caused by the blind signed identity tokens that must be sent as well as the tags.

7.1.1 Comparison to Anonymous Credentials

This section compares the computational complexity of the tagged transaction protocol and anonymous credentials [15]. Anonymous credentials can provide one-show credentials to users that can anonymously establish digital provenance in reseller chains. One-show credentials can also provide global anonymity

revocation if a one-show credential is replayed. I compare the computational complexity of the tagged transaction protocol with the computational complexity of one-show anonymous credentials. Both protocols assume a perfect anonymous communication channel and these complexity figures do not include the complexity for providing this anonymous communication channel. The second set of figures compare the extended tagged transaction protocol that provides revocable anonymity for a tag that is cloned with the one time anonymous credentials with global anonymity revocation. The figures for the tagged transaction protocol and the extended tagged transaction protocol do not include the registration step and assume a single TGC is used. The figures for the anonymous credentials do not include the generation of the pseudonyms that are used to provide anonymity. It is assumed that both protocols are using the same key size.

Table 7.5 uses the following notation:

- x is the complexity for modular exponentiation.
- y is the complexity for modular division.
- r is the number of resellers.

Protocol	Without Anonymity Revocation	With Anonymity Revocation
Tagged Transactions	$y + 19x + r(10x + y)$	$2y + 51x + r(2y + 41x)$
Anonymous Credentials	$r(39x + 6y)$	$r(82x + 10y)$

Table 7.5: Tagged Transaction Protocol vs Anonymous Credentials

Table 7.5 shows the comparison between the computational complexity of the tagged transactions protocol and anonymous credentials. For one reseller, the tagged transaction protocol has a lower complexity than anonymous credentials but the extended tagged transaction protocol has a higher complexity than anonymous credentials with anonymity revocation. Both the tagged transaction protocol and the extended tagged transaction protocol scale more efficiently with the number of resellers. The tagged transaction protocol was designed to provide efficient transferring of tags from one reseller to another. The anonymous credential protocol was designed to provide multiple show credentials with one-show credentials being an extension to the original protocol. For a one-show credential, each reseller must generate the anonymous credential and then prove ownership of it using zero knowledge proofs of knowledge to generate the credential for the next reseller in the chain. The zero knowledge proofs

of knowledge used in anonymous credentials make it more expensive than the tagged transaction protocol when using multiple resellers.

7.1.2 Summary

In this section I have presented the computational and communication complexity of the tagged transaction protocol and the extended tagged transaction protocol. The extended tagged transaction protocol has a higher computational and communication complexity than the tagged transaction protocol due to the extra operations needed to provide revocable anonymity for resellers that clone tags using restricted blind signatures. The details of these complexity results would change if a different signature, encryption, or restricted blind signature scheme were used.

Both the computational and communication complexity of the protocols scale linearly with the total number of TGCs when registering items. This is because the item needs to be registered with every TGC. The complexity scales linearly with the number of resellers in the chain. The complexity also scales quadratically with the threshold number of TGCs.

The anonymous credentials protocol can provide one-show credentials to users that can anonymously establish digital provenance in reseller chains. Anonymous credentials can be used in two different modes, one without revocable anonymity and one that provides revocable anonymity. I have compared the computational complexity of the tagged transaction protocol and the extended tagged transaction protocol to anonymous credentials. The tagged transaction protocol has less computational complexity when compared to anonymous credentials without anonymity revocation and also scales better with the number of resellers. The extended tagged transaction protocol has a higher computational complexity than anonymous credentials with revocable anonymity when only using one reseller but scales better with the number of resellers.

I now consider an example of the computational and communication complexity of the tagged transaction protocol and the extended tagged transaction protocol. I assume there are three resellers, a (4,5) threshold group of TGCs, the output of the hash function is 512 bits, the size of the license is 3096 bytes, and the size of the keys used in the encryption and signature schemes are 2048 bits. Then the computational complexity of the tagged transaction protocol is $5y + 562x$ and the extended tagged transaction protocol is $33y + 1336x$ where y

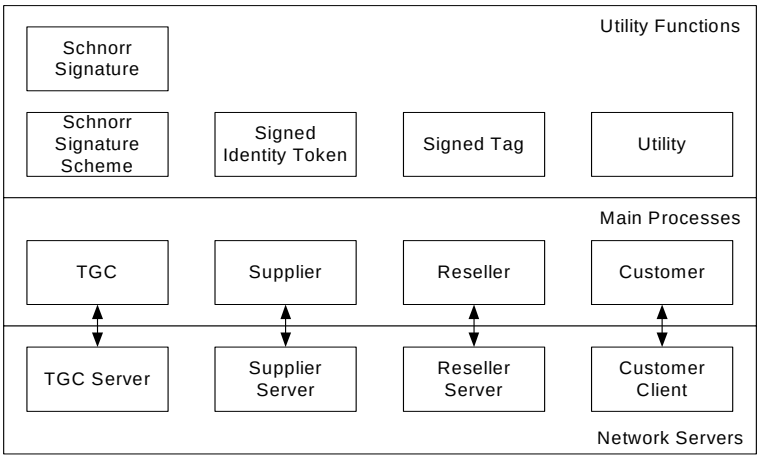

Figure 7.1: Implementation Block Diagram

is the computational complexity of modular division and x is the computational complexity of modular exponentiation. The communication complexity in this example would be 342.1 KB and the size of a tag would be 4.6 KB. For the extended tagged transaction protocol the communication complexity would be 459.7 KB and the size of a tag would be 6.1 KB.

7.2 Implementation Details

To measure the performance of the tagged transaction protocol I have implemented both the tagged transaction protocol and the extended tagged transaction protocol in Java. Figure 7.1 shows a block diagram for the Java implementation. The classes implemented are divided in to the network servers, the main processes, and utility functions.

The utility functions implement any signature or utility functions that are not provided as part of the Java API. The Schnorr signature scheme is not implemented as part of the Java API. I have implemented this as two Java classes. The first class is called SchnorrSignatureScheme that contains all the methods for generating keys, signing data, and verifying signatures. The second class is called SchnorrSignature which contains the information that is needed to send a Schnorr signature. This class is serializable. The class SignedIdentityToken in the utility functions implements the restricted blind signature scheme. The class SignedTag contains the information that is needed to send a signed tag across

the network. This includes any information needed for the extended tagged transaction protocol. This class is serializable. RSA encryption is implemented using the bouncy castle Java cryptography APIs [1] (version 1.45). Random number generation uses the Java SecureRandom class. The SHA-512 hash function is implemented using the Java MessageDigest class.

The main process classes implement the TGC, suppliers, resellers, and customers. Each of these is implemented in a separate class. These classes can make use of the utility classes to create signatures or verify signed items. Keeping the utility classes separate from the main process classes allows the type of signature or encryption to be changed without needing major changes to these main process classes. The customer and reseller class currently share some code as both resellers and customers have to verify signed tags and signed identity tokens. The TGC contains methods to generate, check, and sign tags using the utility classes.

The network servers implement the communication of the various processes using Sockets. Using separate classes for the network servers allows the method of communication to be changed without needing to change the main classes. Currently the network servers also include code to measure the time taken to perform various actions which are written to files to allow the performance tests to take place. The network servers contain the main method for each process which can be configured using command line arguments to use a variable number of TGCs to test the performance of the threshold version of the tagged transaction protocol and a variable number of resellers to test the performance of the protocol with multiple resellers. The command line interface is currently used to specify the address and port of other processes that are needed. The customer and resellers need to know the address and port number of the TGC as well as the address and port number of the supplier or reseller that is adjacent to them in the chain of resellers. The supplier needs to know the address and port number of the TGC.

Tests have been done using the TOR anonymous communication channel using the TOR Linux client[2] (version 2.2.34-2). Connections that requires anonymity connect via a local proxy provided as part of the TOR library. The TOR library retains the same anonymised IP address for 10 minutes before attaining a new anonymised IP address. This means that, in this implementation, the actions of

[1] http://www.bouncycastle.org/java.html

[2] https://www.torproject.org/download/download.html.en

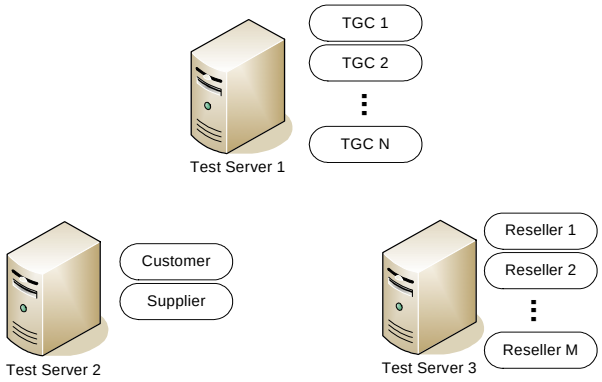

Figure 7.2: Experimental Setup

an individual reseller or supplier are linkable within the 10 minute window.

7.3 Experimental Setup

Figure 7.2 shows the experimental setup for the performance tests. All three test server machines are Dell Optiplex GX780s with an Intel Core 2 Duo E8400 3.0Ghz CPU. Test Server 2 and 3 have 4GB of RAM and Test Server 1 has 16GB of RAM. Test Server 1 runs the group of TGCs that are used. In these experiments this number varies from 1 to 7. Test Server 2 runs a customer and a supplier. Test Server 3 runs any resellers that are used in the protocol. In these experiments this number varies from 1 to 5. The test servers are all connected to the same local network. To measure the performance of the tagged transaction protocol the experiments vary the key size used, the number of TGCs in the group, and the number of resellers between the supplier and the customer. Table 7.6 shows the default values of the parameters.

Parameter	Value
Key Size	1024 bits
Number of TGCs	1
Number of Resellers	1

Table 7.6: Default Experimental Parameters

The performance measurements for registering an item includes the complete time for the supplier to:

1. Generate the identity of the item.

2. Generate a private and public key pair for the item.

3. Encrypt the public key and identity of the item with the public encryption key of the TGC and send this item to the TGC.

4. Receive and check the signed receipt from the TGC.

5. Repeat steps 3 and 4 for all the TGCs in the group.

The performance measurement for the tagged transaction protocol includes the complete time for the following steps:

1. The customer generates a one time key pair for the transaction and sends the one time public key and the identity of the item to the first reseller in the chain.

2. The reseller generates a one time key pair for the transaction and sends the one time public key and the identity of the item to the next reseller in the chain (or the supplier if it is the end of the chain).

3. Step 2 is repeated for each reseller in the chain.

4. The supplier sends a signed purchase request including the one time public key they received and a signed license to the TGC.

5. The TGC checks the signed tag request and generates and returns a signed tag to the supplier.

6. Steps 4 and 5 are repeated for a threshold number of the TGCs in the group.

7. The supplier returns the signed tags it has received to the reseller.

8. The reseller sends the TGC a regeneration request signed using its one time private key containing the public key of the participant before it in the reseller chain. The TGC then signs the new tag and sends it to the reseller.

9. Step 8 is repeated for a threshold number of the TGCs in the group.

10. The customer checks the threshold number of signed tags it has received from the reseller.

The performance measurements for the tagged transaction protocol do not include the registration of the item. The registration step only needs to be completed once for every item and not for every transaction. The performance results represent the worst case performance. Many of the steps in the protocol could be pre-computed. For example, a reseller could pre-purchase a group of tags from the supplier and so not need to complete this step when a customer tries to purchase the item from them.

The performance measurement for the extended tagged transaction protocol includes the complete time for the following steps:

1. The customer completes the withdrawal protocol with the TGC to get a blind signed identity token.
2. Step 1 is repeated for a threshold number of the TGCs in the group.
3. The customer generates a one time key pair for the transaction and sends the blind signed identity tokens, one time public key, and the identity of the item to the first reseller in the chain.
4. The reseller completes the withdrawal protocol with the TGC to get a blind signed identity token.
5. Step 4 is repeated for a threshold number of the TGCs in the group.
6. The reseller generates a one time key pair for the transaction and sends the blind signed identity tokens, one time public key and the identity of the item to the next reseller in the chain (or the supplier if it is the end of the chain).
7. Step 4, 5, and 6 are repeated for each reseller in the chain.
8. The supplier sends a signed purchase request including the blind signed identity token and one time public key they received and a signed license to the TGC.
9. The TGC checks the signed tag request and blind signed identity token and generates and returns a signed tag to the supplier.
10. Steps 8 and 9 are repeated for a threshold number of the TGCs in the group.
11. The supplier returns the signed tags it has received to the reseller.

12. The reseller sends the TGC a regeneration request signed using its one time private key containing the public key of the participant before it in the reseller chain.

13. The reseller and TGC take part in a challenge response protocol to show the reseller owns the blind signed identity token. The TGC then signs the new tag and sends it to the reseller.

14. Steps 12 and 13 are repeated for a threshold number of the TGCs in the group.

15. The customer checks the threshold number of signed tags it has received from the reseller.

The performance measurements for the extended tagged transaction protocol do not include the registration of the item or the opening of an account with the TGCs. Both the registration of the item and the registering of accounts only need to be completed once for every item and not for every transaction. The performance results represent the worst case performance. Many of the steps in the protocol could be pre-computed. For example, a reseller could pre-purchase a group of tags from the supplier and so not need to complete this step when a customer tries to purchase the item from them. Alternately, the blind signed identity tokens could be withdrawn in bulk before the protocol began.

7.4 Experimental Results

This section shows the results of running performance experiments on the implementation of the tagged transaction protocol and the extended tagged transaction protocol. The measurements have been made while all the computers are on the local network as well as using anonymous communication over the TOR network. The graphs done over the local network are all coloured red and the graphs using TOR are all shown in blue. Each run was done 30 times and the averages and standard deviations are shown on the graphs.

7.4.1 Registering Items

Figure 7.3 shows the total time taken to register an item increasing exponentially when the key size increases linearly. This is because increasing the key size lin-

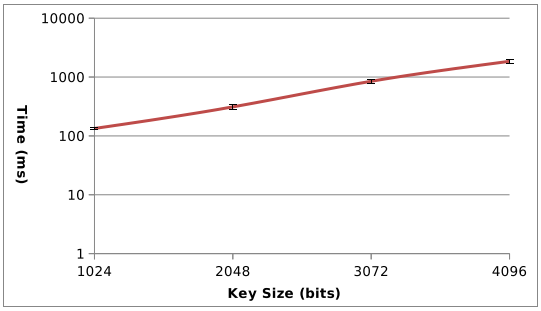

Figure 7.3: Key Size vs Total Time to Register Item

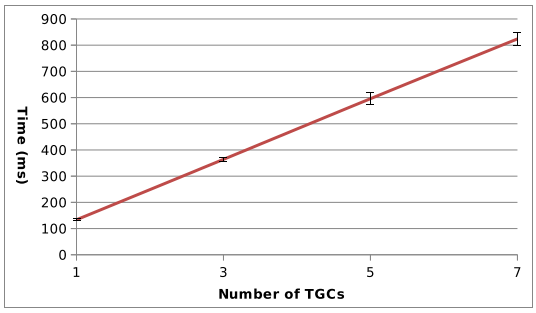

Figure 7.4: Number of TGCs vs Total Time to Register Item

early increases the time taken to complete the modular exponentiation required exponentially.

Increasing the number of total TGCs linearly (note this is the total number of TGCs and not the threshold number) increases the time taken to register the items linearly as shown in Figure 7.4. When a supplier registers an item it has to encrypt and decrypt the message to send for each individual TGC as they all have individual keys. This increases the total time taken linearly.

Using TOR

When using TOR, the time taken to register an item remains almost constant as the key size increases as shown in Figure 7.5. This indicates that the costs of sending the information over the TOR network was the main component of the time taken to register the item and the exponentially increasing time taken to complete the modular exponentiations was not a large factor.

Figure 7.6 shows the time taken to register an item increasing linearly with the total number of TGCs. The largest contributor to the time taken is the data

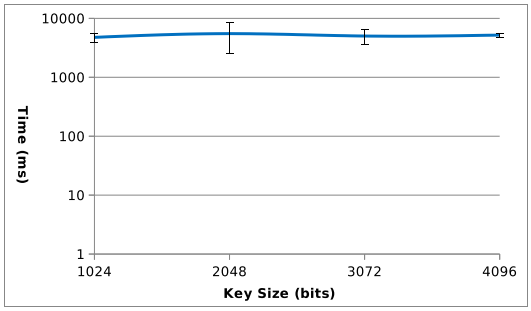

Figure 7.5: Key Size vs Total Time to Register Item using TOR

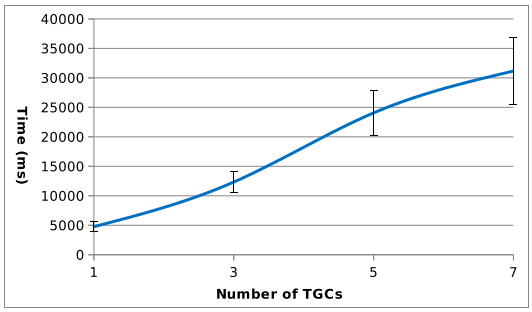

Figure 7.6: Number of TGCs vs Total Time to Register Item using TOR

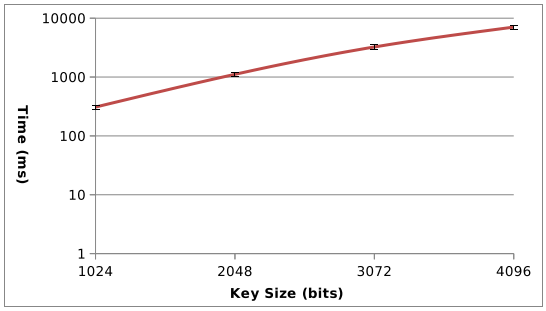

Figure 7.7: Key Size vs Tagged Transaction Total Time

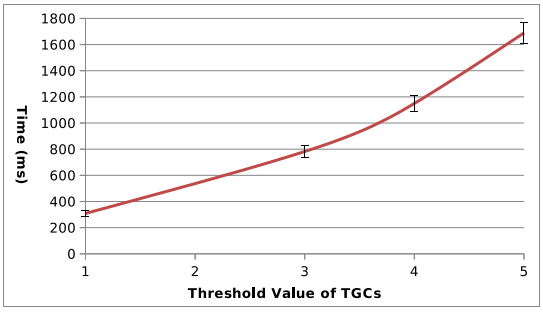

Figure 7.8: Threshold Number of TGCs vs Tagged Transaction Total Time (no data point for the threshold value of 2)

being sent over the TOR network and this increases linearly as the total number of TGCs increases.

7.4.2 Tagged Transaction Protocol

Figure 7.7 shows the time taken to complete the tagged transaction protocol increasing exponentially as the key size increases linearly. This is because as the size of the keys increases linearly, the time taken to do the modular exponentiations and modular division increases exponentially.

The time taken to complete the tagged transaction protocol increases quadratically as the threshold value of TGCs is increased linearly as shown in Figure 7.8. It is worth noting that there is no data point for the threshold number of TGCs of two. Both the computation and communication complexity increase as a function of the square of the threshold value of TGCs (as shown in Table 7.1 and Table 7.3). This is the reason the time taken to complete the tagged transaction protocol shows a quadratic growth as the threshold number of TGCs increases linearly.

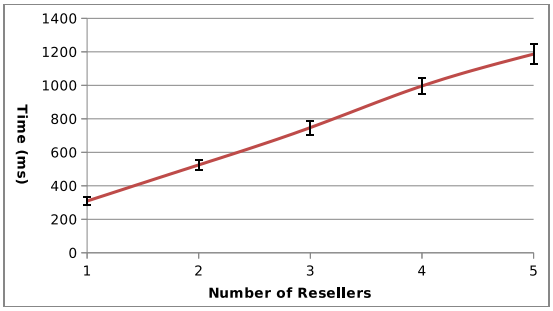

Figure 7.9: Number of Resellers vs Tagged Transaction Total Time

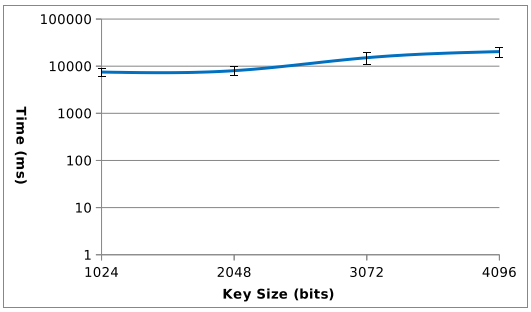

Figure 7.10: Key Size vs Tagged Transaction Total Time using TOR

Figure 7.9 shows the time taken to complete the tagged transaction protocol increasing linearly as the number of resellers increases linearly. Unlike the increase in the threshold number of TGCs, this is only an increase in series. The number of operations performed and the amount of data sent both increase linearly as the number of resellers increases linearly.

Using TOR

Figure 7.10 shows a small exponential increase in time taken for the tagged transaction protocol using TOR as the key size is increased linearly. The shallow slope of this graph as opposed to the steeper slope when increasing the key size without TOR (Figure 7.7) shows that the communication over the TOR network is still the major contributor to the time taken by the tagged transaction protocol.

The time taken to complete the tagged transaction protocol using TOR increases quadratically as the threshold value of TGCs is increased linearly as shown in Figure 7.11. It is worth noting that there is no data point for the threshold number of TGCs of two. Both the computation and communication

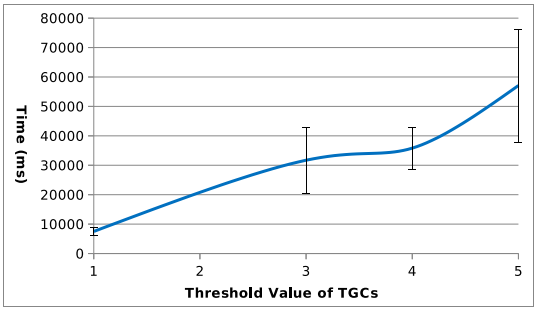

Figure 7.11: Threshold Number of TGCs vs Tagged Transaction Total Time using TOR (no data point for the threshold value of 2)

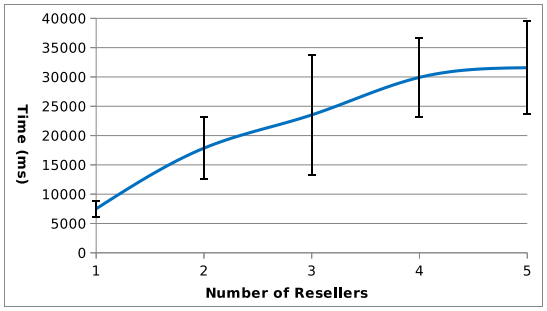

Figure 7.12: Number of Resellers vs Tagged Transaction Total Time using TOR

complexity increase with the square of the threshold number of TGCs causing this quadratic growth in the time taken to complete the tagged transaction protocol.

Figure 7.12 shows the time taken to complete the tagged transaction protocol using TOR increasing linearly as the number of resellers increases. This is because the number of operations and the amount of data sent both increase linearly with the number of resellers. The graph shows a slight curve, but the error bars show that a linear relationship fits this data.

7.4.3 Extended Tagged Transaction Protocol

Figure 7.13 shows the time taken to complete the extended tagged transaction protocol increasing exponentially as the key size increases linearly. This is because as the size of the keys increases linearly, the time taken to do the modular exponentiations and divisions increases exponentially.

The time taken to complete the extended tagged transaction protocol increases quadratically as the threshold value of TGCs is increased linearly as shown in

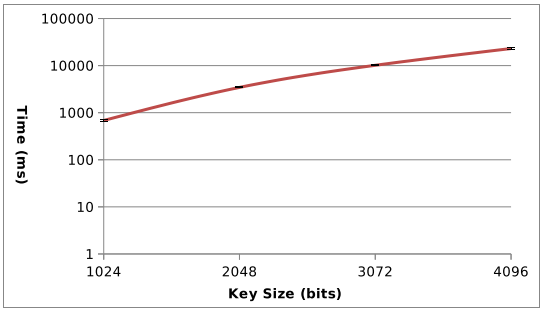

Figure 7.13: Key Size vs Extended Tagged Transaction Total Time

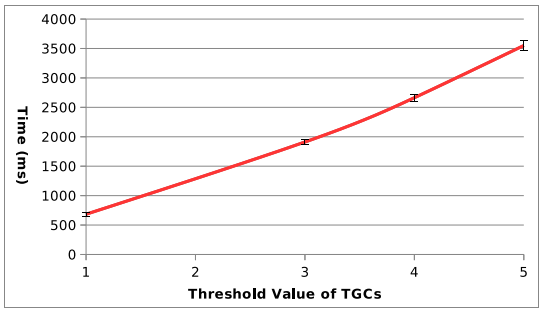

Figure 7.14: Threshold Number of TGCs vs Extended Tagged Transaction Total Time (no data point for the threshold value of 2)

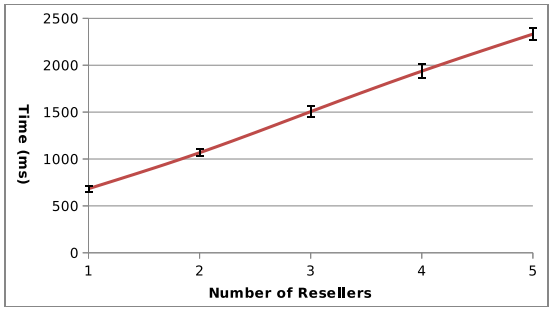

Figure 7.15: Number of Resellers vs Extended Tagged Transaction Total Time

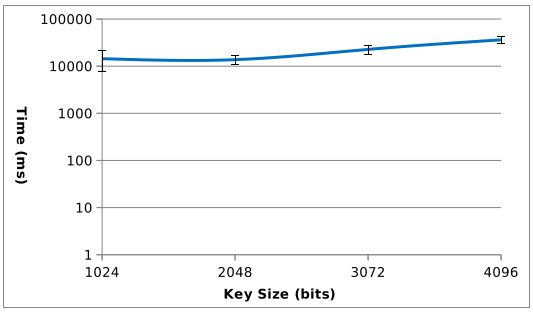

Figure 7.16: Key Size vs Extended Tagged Transaction Total Time using TOR

Figure 7.14. It is worth noting that there is no data point for the threshold number of TGCs of two. Both the computation and communication complexity of the extended tagged transaction protocol increase as a function of the square of the threshold value of TGCs (as shown in Table 7.2 and Table 7.4).

Figure 7.15 shows the time taken to complete the extended tagged transaction protocol increasing linearly as the number of resellers increases linearly. This is because the number of operations performed and the amount of data sent both increase linearly as the number of resellers increases linearly.

Using TOR

Figure 7.16 shows a small exponential increase in time taken for the extended tagged transaction protocol using TOR as the key size is increased linearly. The shallow slope of this graph as opposed to the steeper slope when increasing the key size without TOR (Figure 7.13) shows that the communication over the TOR network is the major contributor to the time taken by the extended tagged transaction protocol.

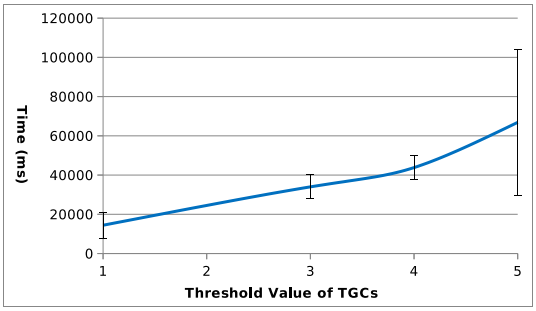

Figure 7.17: Threshold Number of TGCs vs Extended Tagged Transaction Total Time using TOR (no data point for the threshold value of 2)

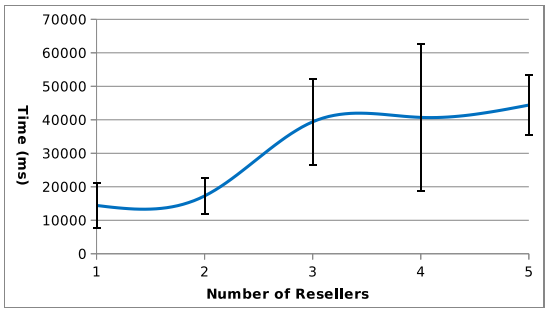

Figure 7.18: Number of Resellers vs Extended Tagged Transaction Total Time using TOR

The time taken to complete the extended tagged transaction protocol using TOR increases quadratically as the threshold value of TGCs is increased linearly as shown in Figure 7.17. It is worth noting that there is no data point for the threshold number of TGCs of two. The computational and communication complexity increases with the square of the threshold number of TGCs causing the quadratic growth in the time taken to complete the extended tagged transaction protocol.

Figure 7.18 shows the time taken to complete the extended tagged transaction protocol increasing with the number of resellers. This graph shows a great deal of variance between different runs with a large standard deviation. As all the resellers have to communicate with the TGC using the TOR network, when there are more resellers and TOR is the major source of variance, then as the resellers are increased the variance of the time taken to complete the protocol will increase.

7.5 Summary

This chapter has shown the performance of the tagged transaction protocol. The complexity results show that the amount of computation to complete both the tagged transaction protocol and the extended tagged transaction protocol increase linearly with the number of resellers in the chain. The computational and communication complexity increase linearly with the square of the threshold number of TGCs.

One-show anonymous credentials can provide anonymous provenance in reseller chains. This chapter shows the computational complexity of the tagged transaction protocol compared to anonymous credentials and the computational complexity of the extended tagged transaction protocol compared to anonymous credentials with revocable global anonymity. With one reseller in the chain, the tagged transaction protocol has a lower complexity than anonymous credentials, but the extended tagged transaction protocol has a higher complexity than anonymous credentials with revocable anonymity. Both the tagged transaction protocol and the extended tagged transaction protocol scale more efficiently when the number of resellers in the chain is increased.

The tagged transaction protocol has been implemented in Java. Using this implementation tests have been conducted to show the effects of varying the parameters of the protocol on the time taken to execute the protocol. Increasing the key size linearly increases the time taken to complete both the tagged transaction protocol and the extended tagged transaction protocol exponentially. Increasing the threshold number of TGCs increase the time taken to compute both the tagged transaction protocol and the extended tagged transaction protocol quadratically. This is due to the communication complexity of the protocol being increased as a function of the square of the threshold number of TGCs. Increasing the number of resellers results in a linear increase in the time taken to complete both the tagged transaction protocol and the extended tagged transaction protocol. The performance results follow what is expected given the complexity results presented earlier in the chapter.

The performance of the tagged transaction protocol has also been shown using the TOR anonymous communication network. Using the TOR network increases the time taken to complete the protocol. The performance tests also show that using TOR introduces a large variance in the time taken to complete the protocol. The more connections made using the TOR network, the greater the variance.

Chapter 8

Provenance in Web Services

This chapter applies the ideas and mechanisms developed for the tagged transaction protocol to providing provenance for web services. In web services, service providers collate and present information from a collection of heterogeneous data sources. These sources can include databases, web applications, and other service providers.

The basic model for web services is shown in Figure 8.1. Consider an example of a web service where users display their photos along with a map showing the location the photo was taken. In this case, the service provider may be using a storage provider and a mapping provider. So in this example, the service provider is providing a service to the user that is a combination of the service provider's own algorithms and the functionality provided by two external service providers.

A service may use information from several different service providers. Depending on the reputation of the service provider being used, users may have differing expectations of the quality of the data as well as the amount they are willing to pay for the service. A service provider that is using a premium service will want to be able to charge more for this service. However, a malicious service provider could claim to be using a premium service when they are using free services provided by a different service provider.

Provenance information for web services refers to the origin and modification history of the data provided as a web service. There are several differences between providing provenance for web services and for reseller transactions. In web services, the service is provided as an ongoing series of service requests and responses as opposed to a one-off item transaction. This means the provenance

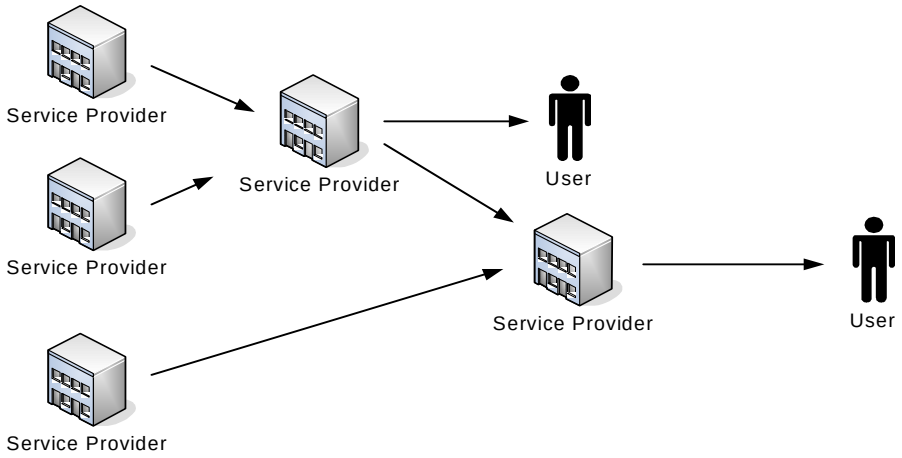

Figure 8.1: Web Services Model

for the service needs to be continually checked through the duration of the service.

When providing provenance for digital items, a digital item will be the same through the reseller chain and have a single supplier. In the web services model, the result of the service may change as various operations are performed by the service providers. Additionally, there is no single supplier for web services. The data can be produced and combined by multiple service providers.

In the reseller model, the participants in the transaction were anonymous and unlinkable to prevent the revelation of potentially commercially sensitive data. In the web services model, these privacy requirements are relaxed. The provenance information provided for a web service lists all the service providers that provided a source of data or modified the data. The identity of a service provider may influence the decision of a user to use the web service.

A subtle problem with providing provenance information for web services is an exclusion attack. In an exclusion attack, the service provider misses out some of the sources of data from the provenance record. In the reseller model, every item has only one supplier and so they cannot be removed from the provenance information without detection but in the web services model a service provider may have many data sources. This chapter presents a discussion on methods to prevent exclusion attacks.

8.1 Domain Model

I define two roles that a participant in a web service can take: service providers and users. A participant may change roles. For example, a user may become a service provider by taking the service provided to it and combining it with some other service or computation to produce a new service.

- Service Providers. Service Providers make use of a heterogeneous collection of data sources to provide a service. A service provider is uniquely identified by a Uniform Resource Identifier (URI). All service providers have a private and public key pair. The public keys of service providers are well known values.
- Users. Users are the end user of the service provided by a service provider.

The terms and actions defined in the Open Provenance Model (OPM) [59] are used to create a model of provenance for web services (similar to the model created for resellers in Section 3.2). The web services model uses the following terms:

- Artifact: An immutable piece of state. In the web services model an artifact is the result of a service. Artifacts are created by service providers and passed to other service providers or users.
- Process: An action or series of actions performed on or caused by an artifact. In the web services model a process is either Generate where a service provider generates some data or TransformCombine where a service provider takes some artifacts and performs some actions on them to create a new artifact.
- Agents: A contextual entity controlling a process. In the web services model all agents are service providers as users do not create provenance information.

The OPM also defines actions, most of which are parametrised by the role of the process or agent that is performing the action. The web services model uses the following actions:

- wascontrolledby(Role): A process is controlled by an agent. The Role parameter is the role the agent is taking in the protocol. SP defines the role of service provider.

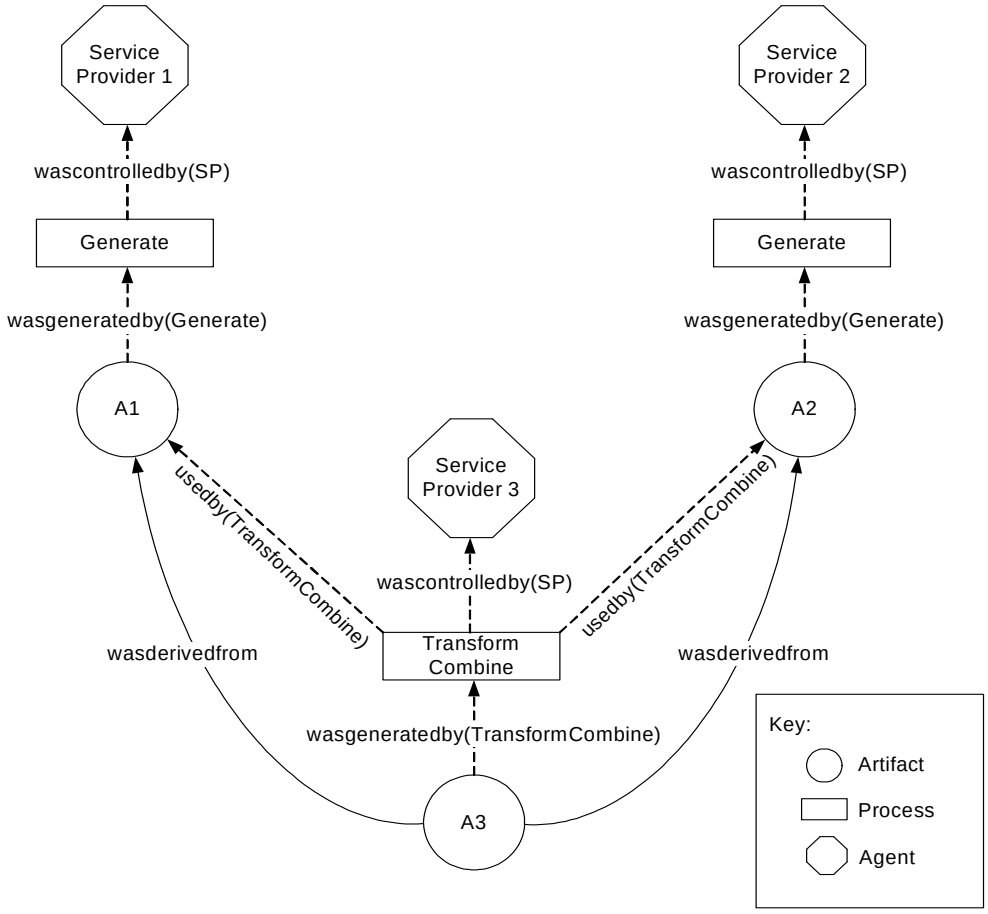

Figure 8.2: Web Services Provenance Graph

- wasgeneratedby(Role): An artifact is generated by a process. The Role parameter is the process that is generating the artifact, in this model this is Generate or TransformCombine.
- usedby(Role): An artifact is used by a process. The Role parameter is the process that uses the artifact, in this model this is the TransformCombine process.
- wasderivedfrom: Artifacts can be derived from other artifacts. There is no role associated with this action.

In the Open Provenance Model, provenance is represented as a provenance graph which is a directed graph where agents are shown as octagons, processes as rectangles, and artifacts as circles. Directed edges in the graph represent an action with the source of the action being the end point of the arrow and the start of the

arrow showing the result of the action. Figure 8.2 shows a provenance graph for a service with two service providers acting as input to a third service provider. The user does not feature in the provenance graph as they are not a source of provenance information. Artifact $A1$ and $A2$ are generated by the Generate process controlled by Service Provider 1 and Service Provider 2 respectively. Artifact $A3$ is generated by the TransformCombine process controlled by Service Provider 3 using $A1$ and $A2$. The graph also shows that $A3$ is derived from $A1$ and $A2$.

8.2 Threat Model

There are many ways a malicious service provider could try and defraud a user with incorrect provenance information. This threat model groups them in to the following categories:

1. Fabrication. The adversary tries to create provenance information for an honest participant in the protocol when it has never obtained the service from the participant.

2. Cloning. The adversary tries to provide multiple users the same provenance information they have obtained from an honest participant in the protocol.

3. Network Sniffing. The adversary replays legitimate provenance information it has seen on the network (possibly intended for a different service provider).

4. Exclusion. The adversary tries to provide information to a user from an honest participant in the protocol without providing providence information.

The service providers are untrusted and may be active adversaries, attempting to create false provenance data, or modify and delete existing provenance data. If a service provider is honest and correctly provides provenance information, then it should not be possible for an adversary to fabricate, clone, network sniff, or exclude the honest service providers provenance information. A service provider may also provide incorrect provenance information to try and discredit another service provider.

While a service provider should provide provenance information for every step in the chain to the user, if a set of service providers on the chain colludes

together they can make it appear as though they are acting as one service provider. We do not consider this an attack as the group is colluding and acting as one party. Side channel attacks are out of the scope of this threat model.

8.3 Provenance Chains

Each time information is created or used by a service provider a provenance tag is created and passed along with the result. The tag is created by the source of the provenance information and signed using the secret key of the source. The tag should contain enough information to recreate the provenance information as shown in the graph in Section 8.1.

A tag is a tuple: $tag = \{A, B, C, D, E\}_{sk_{source}}$. The definitions of the parameters in the tag are:

- $A = source$: the Uniform Resource Identifier (URI) of the source of this provenance tag.
- $B = destination$: the Uniform Resource Identifier (URI) of the destination of this provenance tag.
- $C = serial$: The randomly chosen serial number of this service request. This is created by the user and sent along with the request for service.
- $D = H(data)$: A hash of the service result that is associated with this provenance tag.
- $E = action$: What process was performed on the data, either Generate or TransformCombine.

The tags are signed using the secret key of the source service provider. The notation $\{A\}_{sk_B}$ denotes the message A signed using the key sk_B. It is assumed that all public keys for service providers are well known or discoverable from a certificate authority.

Figure 8.3 shows the process for passing tags between a user and two service providers.

1. The user randomly generates a serial number for this request and sends the request for service to its service provider (Service Provider) along with the serial number. The serial number is to prevent service providers from being able to replay provenance information.

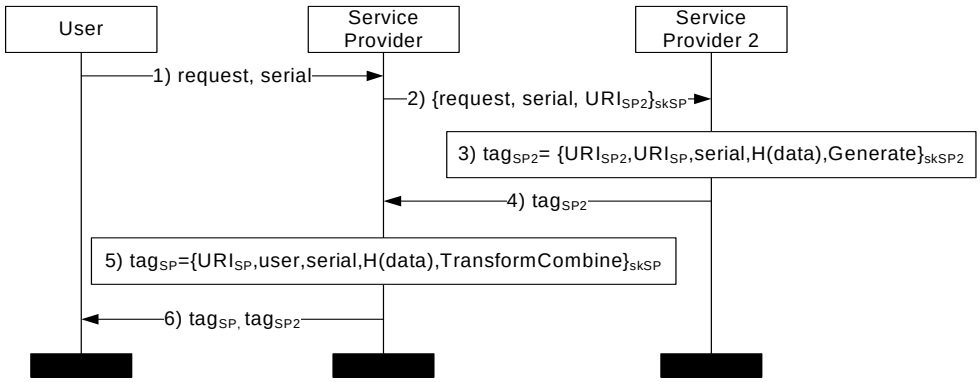

Figure 8.3: Provenance Chains

2. The service provider (Service Provider) generates the request for service from the second service provider (Service Provider 2) and sends it along with the serial number and the URI of the second service provider to the second service provider (Service Provider 2). All the requests sent to service providers are signed using the private key of the sending service provider. The service provider (Service Provider) will send a request to every service provider it uses as input in this step, this example shows one for clarity.

3. The second service provider (Service Provider 2) generates a provenance tag to return to the first service provider. The tag contains the source and destination, the serial number, a hash of the data returned, and the action taken, in this case Generate. The tag is then signed using its secret key sk_{SP2}.

4. The signed provenance tag is returned to the first service provider.

5. The service provider creates a provenance tag to send to the user. The tag contains the source and destination, the serial number, a hash of the data to return to the user, and the action TransformCombine. The service provider then signs this tag using its secret key sk_{SP}.

6. The service provider (Service Provider) sends the user the two signed tags.

The user checks that all provenance information contains the correct serial number, a tag from all registered inputs, and an unbroken chain of provenance information.

8.4 Analysis

The protocol for providing provenance for web services is now compared to our requirements discussed in Section 8.1 and Section 8.2. The analysis examines the protocol in terms of completeness requirements to check that it contains enough provenance information to recreate the provenance graph. A security analysis is then completed to check that incorrect provenance information can be detected in the presence of malicious service providers.

8.4.1 Completeness

A protocol that has complete provenance information should allow the creation of a provenance graph. Completeness is shown by considering the relationships shown in the open provenance model and how the provenance tags show these relationships.

- Artifact: identified by the hash value of the artifact in the provenance tag.
- Process: identified by the process name (either Generate or TransformCombine) in the provenance tag.
- Agent: identified by the source of the provenance tag.
- wascontrolledby(role): is represented by the source of the provenance tag and the role is identified by the process name.
- wasgeneratedby(role): is represented by the hash of the artifact and the role by the process name.
- usedby(role): is represented by the set of tags with the destination set as the controller for the operation of the role.
- wasderivedfrom: can be generated using the source and destination values of the provenance tags and the hash values of the artifacts. wasderivedfrom may be over approximated when there are multiple inputs and outputs for a service provider for a single service request. In this case, the tags show that the output artifacts were derived from the input artifacts, but not what individual input artifacts an output artifact was derived from.

8.4.2 Security Analysis

Service providers may provide incorrect provenance information or respond incorrectly to audit requests. This analysis assumes a signature scheme that has provable security against existential forgeries under adaptive chosen message attacks in the random oracle model such as PSS RSA [7], the triplet El-Gamal scheme [74], or the Schnorr signature scheme [83]. If the signature scheme is secure against existential forgeries, then given a public key pk it is infeasible to forge a pair (m, σ) where σ is a valid signature on m using the secret key corresponding to the public key pk. It is assumed that serial numbers are chosen at random from a large set and that the chance of two users choosing the same serial number is negligible. Side channel attacks or the possibility of an adversary gaining access to the secret key of an honest participant are not considered.

Fabrication

If the adversary is able to construct provenance information from an honest participant in the protocol that they have never used then they must create a tag signed using the secret key for the participant. This adversary can then be used to break the signature scheme. The adversary is given as input the public key of the participant pk. The adversary will then produce a valid provenance tag $\{tag\}_{sk}$. This creates a valid message signature pair for pk, $(m = tag, \sigma = \{tag\}_{sk})$.

Cloning

If the adversary is able to clone provenance information from an honest participant then the adversary must modify a valid signed tag with a new serial number. If the adversary can complete this modification, we can use the adversary to break the signature scheme. The adversary is given as input the public key of the honest participant and the signed tag to clone $\{tag_{original}\}_{sk}$. The adversary will then produce a new tag that has a different serial number denoted $\{tag_{new}\}_{sk}$. This creates a valid message signature pair for pk, $(m = tag_{new}, \sigma = \{tag_{new}\}_{sk})$.

Network Sniffing

If the adversary is able to use network sniffing to claim ownership of some provenance information then, similar to a cloning attack, the adversary will have to alter the tag to have the correct serial number. The adversary is given as

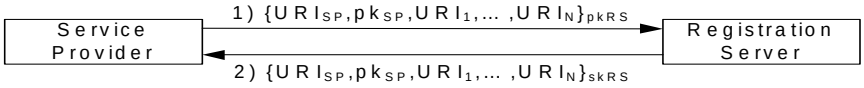

Figure 8.4: Service Provider Registration

input the public key of the honest participant and the tag they have sniffed $\{tag_{original}\}_{sk}$. The adversary will then produce a new tag that has a different serial number denoted $\{tag_{new}\}_{sk}$. This creates a valid message signature pair for pk, $(m = tag_{new}, \sigma = \{tag_{new}\}_{sk})$.

8.5 Preventing Exclusion Attacks

To prevent exclusion attacks, a service provider registers the URIs of service providers it uses as input with a third party called a Registration Server which records these details and is queried by the users to discover the provenance information it should be receiving from a service. All service providers register with the registration server even if they have no input services. The auditing process presented in this section assumes that all input service providers are used for every service request by the service provider.

8.5.1 Service Provider Registration

Figure 8.4 shows the process used by a service provider to register their service with the registration server. The service provider submits a record with their URI URI_{SP}, their public key pk_{SP} and the list of URIs of all service providers they use as inputs $URI_1, ..., URI_N$. This record is encrypted with the public key of the registration server to prevent it being modified by a third party. The registration server then returns a signed receipt. The inputs that a service provider uses may also change over time. In this case the service provider should re-register the service with the new inputs.

8.5.2 User Requesting Service Provider Data

Figure 8.5 shows a user requesting the data on a specific service provider from the registration server. The user sends a request with the URI of the service provider URI_{SP} to the registration server. The registration server then returns a signed

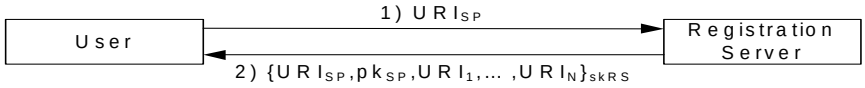

Figure 8.5: User Requesting Service Provider Data

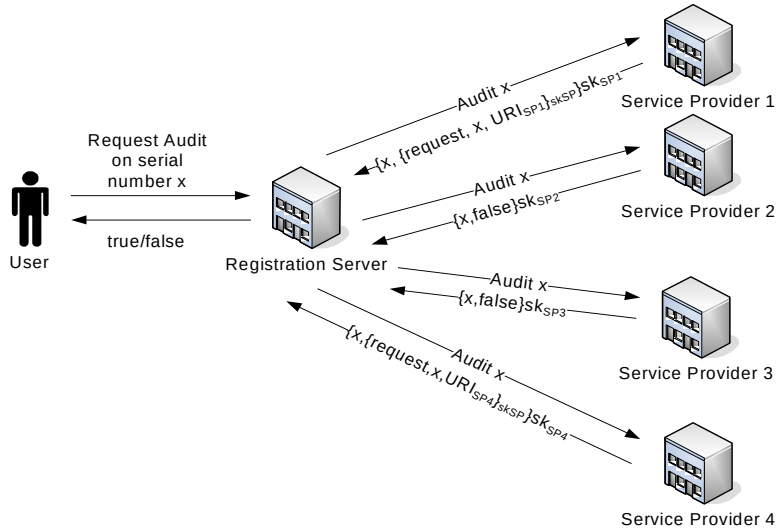

Figure 8.6: Auditing Registration Information

message with the list of input service providers registered by the service provider identified by URI_{SP}. The user will not need to perform this action every time it uses the service, it will only need to periodically check for changes to the inputs to the service provider.

8.5.3 Auditing Registration Information

A service provider may submit incorrect information by excluding some of the service providers it uses to provide its service, thereby hiding a source of data. A user can request an audit for a particular query it has done. The user will submit the serial number it used in the request to the registration server which carries out checks to confirm the registration information provided by the service provider.

Figure 8.6 shows the process for requesting the audit. The registration server broadcasts an audit request with the serial number provided by the user. Service providers then return a signed message containing either false and the serial number if they did not take part in the request, or the the signed request that was

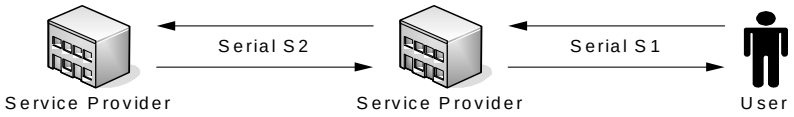

Figure 8.7: Example Exclusion Attack

sent to them when they first responded to the service $\{request, serial, URI\}_{sk_{SP}}$. The service providers return the signed request so that a dishonest service provider cannot frame another service provider in an exclusion attack. These messages are put together by the registration server to check the information it has registered for a service provider.

A broadcast request has to be sent to all service providers, but users do not request an audit for every service request. The registration server could also return a time stamp of the last audit time for the service provider with the information it returns to the user.

If the registration server is acting honestly, then it will send the audit message to all service providers that have registered. All honest participants will have registered with the registration server and so will receive the audit message and respond to it. The registration server can then check the information that is returned to confirm the registration information that has been provided by the service provider.

If the registration server is not trusted and may act maliciously, then some mechanism is needed to verify the actions of the registration server. This analysis briefly discusses two possible mechanisms for implementing a verifiable registration server: a public bulletin board and the use of a group of registration servers. The registration server could publish all its actions to a public bulletin board. Users could then check that the audits they requested have been completed and responders to the audit message can check that their response is correctly recorded. A second option is to use a group of registration servers where a threshold value of the group is required to sign values sent to the user. As long as less than this threshold value of registration servers are acting correctly the user can be confident of receiving correct responses to its queries and audits.

One method for a malicious web service to carry out an exclusion attack in this model is for the service provider to change the serial number during the service. The user sends a web service request with a serial number $S1$ to a service provider. The service provider then creates a new web service request with a new

serial number $S2$ to send to the service provider. When the audit request is made with the serial number $S1$, the service provider that received the request with the serial number $S2$ will return false even if it is acting honestly. This attack is shown in Figure 8.7.

To prevent this kind of exclusion attack, service providers need to be prevented or discouraged from being able to generate a serial number. Suppose the registration server is the only party able to generate serial numbers. An honest service provider will only provide the service and provenance information if the service request has a valid serial number signed by the registration server and users will need to apply to the registration server for valid serial numbers.

To discourage dishonest service providers from generating a serial number a participant that requests a serial number can be required to solve a computational challenge or captcha. A user will then need to solve this computational challenge for every service request they make. This punishes service providers that try to provide web services that exclude their input services by the loss of computation that they require to generate serial numbers. While users will also need complete the computational challenge, it is reasonable to assume that a service provider will have many users and will need to contribute more computational time for an exclusion attack.

A second option would be to charge a participant that generates a serial number a micro-payment. This will mean that an honest user will need to pay a micro-payment every time they request a service. Again it is reasonable to assume that a service provider will need to generate more serial numbers than the users resulting in a significant financial penalty. These micro-payments can be used to fund the resources required for the registration servers. If the web service is non-interactive then a captcha will not be possible and a computational challenge or micro-payment will be necessary.

8.6 Summary

This chapter has applied the techniques and ideas from the tagged transaction protocol to providing provenance for web services. There are several differences between web services and static items: a web service is provided as an ongoing series of requests and responses as opposed to a single item, a web service may dynamically change the service providers they are using, and web services

do not have the same privacy requirements as reseller transactions. A model of provenance for web services has been constructed using the definitions and actions of the Open Provenance Model as well as a list of potential attacks.

A protocol for providing provenance for web services has been constructed using tags that include the source and destination address, a one time serial number, a hash of the data, and what action was performed on the data. These provenance tags are then passed from service provider to service provider before being sent to the user. The user can check the tags contain the correct provenance information. An analysis of the protocol has been completed showing that the protocol provides enough information to reconstruct the provenance graph and that it is secure against fabrication, cloning, and network sniffing attacks.

Exclusion attacks on provenance for web services involve a malicious service provider missing some of the honest input service providers from the provenance information. A registration server can be used to audit the inputs to a web service. To prevent a malicious web service from generating new serial numbers the use of computational puzzles and micropayments has been discussed.

The work presented in this chapter on provenance for web services has many avenues for future work. Future work includes: a more in depth security analysis, more work on preventing exclusion attacks, and measurements on the complexity and performance of the protocol.

Chapter 9

Conclusions and Future Work

This thesis has addressed the problem of anonymously establishing provenance for digital items in reseller chains. Increasingly digital items are purchased online through resellers providing many benefits such as 24-hour access to items and never having stores sell out of items. However, some properties of bricks and mortar stores have been lost when purchasing items through online resellers such as the ability to resell items once customers have finished with them.

Adversaries have many ways to try and defraud suppliers and customers by trying to pass themselves of as the supplier, fabricate licenses for items, clone licenses for items, and sniff a valid license on the network and claim it as their own. There are also privacy issues where a reseller may want to keep their supply chain anonymous and not allow tracking of their transactions.

Secure provenance has been formally defined in this thesis as *anonymously establishing provenance of digital items in reseller chains in the presence of active adversaries*. The tagged transaction protocol developed in this thesis provides secure provenance. The tagged transaction protocol uses a third party called a Tag Generation Centre (TGC) and data structures called tags to provide secure provenance. A second protocol called the extended tagged transaction protocol has also been developed that provides secure provenance while allowing revocable anonymity if a reseller tries to clone a tag. The extended tagged transaction protocol uses ideas from digital cash to provide revocable anonymity for resellers that try to clone tags while preserving the privacy of participants that do not clone tags. Methods have been presented to verify the actions of the TGC to remove the reliance on the TGC acting as a trusted third party.

Table 9.1 shows the properties of the tagged transactions protocol, the ex-

Protocol	Model	R1	R2	R3	R4	R5	R6	R7	R8
Tagged Transactions	Digital Items	✓	✓	✓	✓	✓	✓	✓	✓
Extended Tagged Transactions	Digital Items	✓	✓	✓	✓	✓	✓	✓	✓
Web Services	Web Services	✓	✓	✓			✓	✓	

Table 9.1: Properties of Developed Protocols

tended tagged transaction protocol, and the protocol for providing provenance in web services compared to the requirements listed in Section 1.2. These requirements are briefly restated here as:

- R1: Establish Provenance of Digital Items/Web Services
- R2: Support Multiple Resellers/Service Providers
- R3: Support Customer Reselling
- R4: Provide Anonymity
- R5: Unlinkability of Actions
- R6: Efficient for Low Powered Devices
- R7: Agnostic to Deliver Method
- R8: Allow Bulk Buying

Both the tagged transaction protocol and the extended tagged transaction protocol achieve all the requirements. The protocol for providing provenance in web services does not provide anonymity or unlinkability as these requirements are not required in the model of web services provenance in this work. In the tagged transaction protocol and the extended tagged transaction protocol the complexity for a customer increases linearly with the threshold number of TGCs and is constant for the number of resellers. The complexity of the protocol for providing provenance for web services scales linearly with the number of service providers that are contributing to the service.

A thorough security analysis has been performed on both the tagged transaction protocol and the extended tagged transaction protocol. This security analysis uses a provable security style argument of contradiction. If an adversary can break the properties of secure provenance for either the tagged transaction

protocol or the extended tagged transaction protocol then this adversary can be used to solve a problem thought to be hard. The tagged transaction protocol and the extended tagged transaction protocol have been shown to be secure against spoofing, fabrication, cloning, and network sniffing attacks using the Casper compiler and the Failures Divergences Refinement (FDR) state based model checker. The FDR model checker takes a description of the protocol and checks all possible states of the protocol to show that it does not violate a specification. The adversary in the FDR model checker has complete control of the network and can add, drop, intercept, and fake messages.

A computational complexity and communication complexity analysis has been completed on the tagged transaction protocol and the extended tagged transaction protocol. These results show that the both the computational and communication complexity scale linearly with the number of resellers. When multiple TGCs are used the computational and communication complexity scale linearly with the square of the number of TGCs. A comparison has been done between the computational complexity of the tagged transaction protocol and the extended tagged transaction protocol and a protocol called anonymous credentials. Anonymous credentials can also be used to provide secure provenance for digital items in reseller chains. This analysis shows that the tagged transaction protocol has less computational complexity than anonymous credentials, and scales better with the number of resellers. The extended tagged transaction protocol has more computational complexity than anonymous credentials when one reseller is used, but scales better with the number of resellers.

The tagged transaction protocol and extended tagged transaction protocol have been implemented in Java. Experimental tests have examined the effect of varying the key size, threshold value of TGCs when using a multiple party TGC distribution, number of resellers, and degree of anonymity on the time taken to complete the protocols. These tests show that the time taken to complete the protocols increases exponentially with the key size, and linearly with the number of resellers. Implementing threshold-based trust for the TGCs means that whenever an additional TGC is added the number of tags that each TGC must receive and check each time a tag is generated by a reseller is increased. This causes the time taken to increase quadratically with the threshold number of TGCs. Tests were conducted using the TOR communication network which showed that most of the time taken was in the communication over TOR and a large variance in the time taken to send data over the TOR network.

The ideas and mechanisms from the tagged transaction protocol have been applied to providing provenance for web services. The web services model has several differences to the digital item model. Web services provide a service that is on going and do not have the same privacy requirements as reseller transactions. The tag data structure has been modified to provide provenance for web services. A basic analysis of the protocol for providing provenance in web services has been completed showing that the protocol provides complete provenance information and security against spoofing, cloning, and network sniffing attacks. This work defines an exclusion attack where the service provider misses out some of the sources of data from the provenance record and a discussion of methods to prevent exclusion attacks.

9.1 Contributions

The main contributions in this thesis are:

1. Formalising the requirements and algorithms for a protocol for anonymously providing provenance in reseller chains. Based on the possible attacks, this results in a formal definition of secure provenance.

2. Development of the tagged transaction protocol and the extended tagged transaction protocol. Both protocols use tags and a third party called the Tag Generation Centre (TGC) to anonymously provide provenance in reseller chains.

3. Security analysis of the tagged transaction protocol using arguments by contradiction and model checking.

4. Implementation and performance measurements of the tagged transaction protocol showing the effects of the choice of distribution and anonymous communication channel used.

5. Application of the ideas from the tagged transaction protocol to provide secure provenance in web services. This includes the definition and discussion of methods to prevent exclusion attacks.

9.2 Future Work

There are several areas in this thesis that would provide interesting avenues for future work. For the tagged transaction protocol and extended tagged transaction protocol extending the FDR modelling results to include anonymity and unlinkability would provide a more complete security analysis. The FDR model checker has been used in the past to model anonymity properties and it would be good to apply these techniques to these protocols.

The experimental results for the tagged transaction protocol and the extended tagged transaction protocol currently have only been completed with all the participants on the local network or communicating through TOR. Extending these results by distributing the parties in global locations would provide a more realistic test bed for the protocol. Distributing the parties would also provide a base line for the results using the TOR network and show whether the variance and extra time are caused by the TOR network or the communication over the global network.

An opportunity exists to do a more in depth analysis of provenance in web services through the application of the FDR model checker as used with the tagged transaction protocol.

Currently solving exclusion attacks on provenance in web services is an open problem. While this work has suggested some mechanisms to discourage service providers from doing exclusion attacks, a mechanism to detect or prevent exclusion attacks would be interesting to explore.

In the current design of the protocol for providing provenance in web services, when there are multiple inputs and outputs for a service provider, the protocol shows that the outputs were derived from the inputs but not what individual input an output was derived from. Changing the protocol to show the individual inputs that resulted in an output would provide a more fine-grained view of the provenance information.

Appendix A

CSP Models

This appendix details the CSP models that are created to check the security properties of the tagged transaction protocol. The second section shows the models for the extended tagged transaction protocol.

A.1 Tagged Transaction Protocol

The tagged transaction protocol has three CSP models: registration, the supplier generating a tag, and a reseller generating a tag.

A.1.1 Registration

In the registration phase, the supplier is registering the identity of a new item with the TGC along with a public key for the item.

Sets

$$Agents = \{S, T, A\}$$

$$Items = \{item\}$$

$$ItemKeys = \{itemkey, intruderkey\}$$

$$Key = \{PK_S, SK_S, PK_T, SK_T, PK_A, SK_A\}$$

The set $Agents$ is composed of the participants in the registration phase and the processes in the CSP model. The values S, T, and A represent the supplier, TGC, and the adversary respectively. The set $Items$ is composed of the items the supplier wants to register. The set $ItemKeys$ is composed of the public keys that

are associated with the item. The set Key is the set of public and private keys associated with each Agent where PK is the public key and SK is the private key.

Messages

The two messages that are part of the registration phase of the tagged transaction protocol are denoted $MSG1$ and $MSG2$:

$$MSG1 \mathrel{\widehat{=}} \{Msg1.a.b.Encrypt.k.item.itemkey \mid a \in Agents, b \in Agents, k \in Key, item \in Items, itemkey \in ItemKeys\}$$

$$MSG2 \mathrel{\widehat{=}} \{Msg2.a.b.Encrypt.k.item.itemkey \mid a \in Agents, b \in Agents, k \in Key, item \in Items, itemkey \in ItemKeys\}$$

The set of all messages in this phase of the protocol is defined as:

$$MSG \mathrel{\widehat{=}} MSG1 \cup MSG2$$

The channels defined in this system are $send$, $receive$, $intercept$, $fake$, $running$, and $commit$. The channels $send$ and $receive$ are used by the processes to exchange messages. The channels $intercept$ and $fake$ are used by the adversary to intercept messages and insert new messages. The channels $running$ and $commit$ are used to represent the state of the processes. Formally in CSP the channels are defined as:
channel $send, receive, intercept, fake : MSG$
channel $running, commit$

Processes

The supplier is defined as:

$$SUPPLIER(item, itemkey, PK_T) \mathrel{\widehat{=}} send.Msg1.S.T.Encrypt.PK_T.item.itemkey \rightarrow running.S.T.item.itemkey \rightarrow receive.Msg2.T.S.Encrypt.SK_T.item.itemkey \rightarrow SKIP$$

and the TGC is defined as:

$TGC(SK_T, PK_T) \mathrel{\widehat{=}}$
$receive.Msg1.S.T.Encrypt.PK_T.item.itemkey \rightarrow$
$send.Msg2.T.S.Encrypt.SK_T.item.itemkey \rightarrow$
$commit.T.S.item.itemkey \rightarrow SKIP$

A renaming is applied to the definitions of the supplier and the TGC to allow the adversary to intercept and fake messages on the communication channels. The renaming allows the input to come from either the channel $receive$ or from the channel $fake$ and the output to go to either $send$ or $intercept$. The supplier and TGC cannot tell whether a message has been sent from the other party or from the adversary.

$SUPPLIER_0 \mathrel{\widehat{=}} SUPPLIER(item, itemkey, PK_T)$
$[[send.Msg1 \leftarrow send.Msg1, send.Msg1 \leftarrow intercept.Msg1,$
$receive.Msg2 \leftarrow receive.Msg2, receive.Msg2 \leftarrow fake.Msg2]]$

$TGC_0 \mathrel{\widehat{=}} TGC(SK_T, PK_T)$
$[[receive.Msg1 \leftarrow receive.Msg1, receive.Msg1 \leftarrow fake.Msg1,$
$send.Msg2 \leftarrow send.Msg2, send.Msg2 \leftarrow intercept.Msg2]]$

The Adversary

The adversary is parametrised by the information it knows. In this process the adversary is parametrised by $m1s$ which is the set of all message 1s the adversary knows, $m2s$ which is the set of all message 2s the adversary knows, i which is the set of all $Items$ the adversary knows, and ik which is the set of all $ItemKeys$ the adversary knows. If the adversary knows the key to decrypt a message it learns the contents of the message, otherwise the adversary learns the encrypted content of the message. The adversary can fake new messages using any values it has learnt, it can also fake messages by replaying the encrypted content of a message it has seen. The CSP definition is:

$I(m1s,m2s, i, ik) \mathrel{\widehat{=}}$
$receive.Msg1?a.b.Encrypt.k.i'.ik' \rightarrow$
if $k \in \{SK_S, SK_T, SK_A, PK_A\}$ then $I(m1s, m2s, i \cup \{i'\}, ik \cup \{ik'\})$
else $I(m1s \cup \{Encrypt.k.i'.ik'\}, m2s, i, ik)$

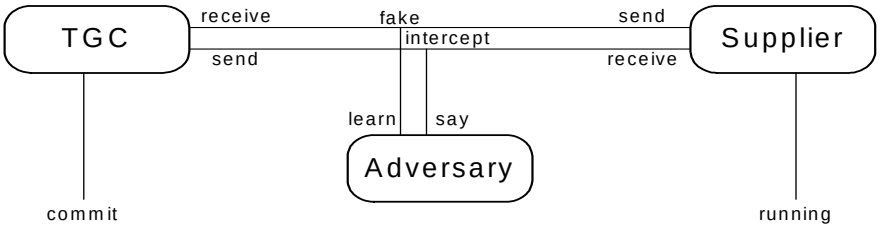

Figure A.1: CSP Network Model: Registration

$\Box\ intercept.Msg1?a.b.Encrypt.k.i'.ik' \rightarrow$
if $k \in \{SK_S, SK_T, SK_A, PK_A\}$ then $I(m1s, m2s, i \cup \{i'\}, ik \cup \{ik'\})$
else $I(m1s \cup \{Encrypt.k.i'.ik'\}, m2s, i, ik)$
$\Box\ receive.Msg2?b.a.Encrypt.k.i'.ik' \rightarrow$
if $k \in \{SK_S, SK_T, SK_A, PK_A\}$ then $I(m1s, m2s, i \cup \{i'\}, ik \cup \{ik'\})$
else $I(m1s, m2s \cup \{Encrypt.k.i'.ik'\}, i, ik)$
$\Box\ intercept.Msg2?b.a.Encrypt.k.i'.ik' \rightarrow$
if $k \in \{SK_S, SK_T, SK_A, PK_A\}$ then $I(m1s, m2s, i \cup \{i'\}, ik \cup \{ik'\})$
else $I(m1s, m2s \cup \{Encrypt.k.i'.ik'\}, i, ik)$
$fake.Msg1?a.b?m : m1s \rightarrow I(m1s, m2s, i, ik)$
$fake.Msg2?b.a?m : m2s \rightarrow I(m1s, m2s, i, ik)$
$fake.Msg1?a.b!Encrypt?k?i' : i?ik' : ik \rightarrow I(m1s, m2s, i, ik)$
$fake.Msg2?b.a!Encrypt?k?i' : i?ik' : ik \rightarrow I(m1s, m2s, i, ik)$

This model considers an adversary with the initial knowledge of $intruderkey$:

$$INTRUDER \mathrel{\widehat{=}} I(\{\}, \{\}, \{\}, \{intruderkey\})$$

Verification

The model of the protocol is now put together with the intruder:

$$AGENTS \mathrel{\widehat{=}} SUPPLIER_0 |[\{|send, receive|\}]| TGC_0$$

$$SYSTEM \mathrel{\widehat{=}} AGENTS |[\{|send, receive, fake, intercept|\}]| INTRUDER$$

Figure A.1 illustrates the processes and channels in the model of the protocol. The supplier and the TGC process have access to the channels $send$ and $receive$. The supplier also has access to the channel $running$ and the TGC has access to the channel $commit$. The adversary has access to the channels $fake$ and $intercept$.

FDR takes as input an implementation of the protocol and a specification and checks that the implementation refines the specification. The registration step has to prevent a spoofing attack where an adversary registers the item before the supplier. Informally, if the TGC receives a registration message for an item, the supplier must have sent the registration message. In CSP the specification is defined as:

$$SPEC_0 \mathrel{\widehat{=}} running.S.T.item.itemkey \rightarrow commit.T?S.item.itemkey \rightarrow SPEC$$

$$SPEC_1 \mathrel{\widehat{=}} \{|running.S.T.item.itemkey, commit.T.S.item.itemkey|\}$$

$$SPEC \mathrel{\widehat{=}} SPEC_0 \left|\left|\left| RUN(\Sigma \setminus SPEC_1)\right.\right.\right.$$

The term Σ is the set of all possible events. The FDR model checker checks $SPEC \sqsubseteq_{\mathrm{T}} SYSTEM$ and returns TRUE after 10 states with 20 transitions.

A.1.2 Supplier Generating Tag

In this phase, the supplier generates a tag for a reseller with the TGC.

Sets

$$Agents = \{S, T, R, A\}$$

$$Items = \{item\}$$

$$Tags = \{pktagreseller, pktagwrong\}$$

$$ItemKeys = \{PKItem_{item}, SKItem_{item}\}$$

$$Key = \{PK_S, SK_S, PK_T, SK_T, PK_C, SK_C, PK_A, SK_A\}$$

The set $Agents$ is composed of the participants involved when the supplier generates the tag with the TGC and the processes in the CSP model. The values S, T, R, and A represent the supplier, TGC, reseller, and adversary respectively. The set $Items$ is composed of the items that reseller wants to purchase. The set $Tags$ is composed of the one time keys that are in the tags. The set $ItemKeys$ is composed of the private and public keys that are associated with the item where $PKItem$ is the public key and $SKItem$ is the private key. The set Key is the set of public and private keys associated with each Agent where PK is the public key and SK is the private key.

Messages

The four messages that are part of the registration phase of the tagged transaction protocol denoted $MSG1$, $MSG2$, $MSG3$, and $MSG4$ are:

$$MSG1 \mathrel{\widehat{=}} \{Msg1.a.b.tag \mid a \in Agents, b \in Agents, tag \in Tags\}$$

$$MSG2 \mathrel{\widehat{=}} \{Msg2.a.b.Encrypt.k.tag \mid a \in Agents, b \in Agents, k \in ItemKeys, tag \in Tags\}$$

$$MSG3 \mathrel{\widehat{=}} \{Msg3.a.b.Encrypt.k.itemkey.tag \mid a \in Agents, b \in Agents, k \in Key, itemkey \in ItemKeys, tag \in Tags\}$$

$$MSG4 \mathrel{\widehat{=}} \{Msg4.a.b.Encrypt.k.itemkey.tag \mid a \in Agents, b \in Agents, k \in Key, itemkey \in ItemKeys, tag \in Tags\}$$

The set of all messages in this phase of the protocol is defined as:

$$MSG \mathrel{\widehat{=}} MSG1 \cup MSG2 \cup MSG3 \cup MSG4$$

The channels defined in this system are $send$, $receive$, $intercept$, $fake$, $running$, and $commit$. The channels $send$ and $receive$ are used by the processes to exchange messages. The channels $intercept$, and $fake$ are used by the adversary to intercept messages and insert new messages. The channels $running$ and $commit$ are used to represent the state of the processes. Formally in CSP the channels are defined as:
channel $send, receive, intercept, fake : MSG$
channel $running, commit$

Processes

The reseller is defined as:

$$RESELLER(item, pktagreseller, PK_T) \mathrel{\widehat{=}}$$
$$send.Msg1.R.S.pktagreseller \rightarrow$$
$$receive.Msg4.S.R.Encrypt.SK_T.PKItem_{item}.pktagreseller \rightarrow$$

$commit.R.S.pktagcustomer \rightarrow SKIP$

The supplier is defined as:

$$SUPPLIER(item, SKItem_{item}, PKItem_{item}, PK_T) \mathrel{\widehat{=}}$$
$$receive.Msg1.R.S.pktagreseller \rightarrow$$
$$running.R.S.pktagreseller \rightarrow$$
$$send.Msg2.S.T.Encrypt.SKItem_{item}.pktagreseller \rightarrow$$
$$receive.Msg3.T.S.Encrypt.SK_T.PKItem(item).pktagreseller \rightarrow$$
$$send.Msg4.S.R.Encrypt.SK_T.PKItem(item).pktagreseller \rightarrow SKIP$$

and the TGC is defined as:

$$TGC(SK_T, PK_T) \mathrel{\widehat{=}}$$
$$receive.Msg2.S.T.Encrypt.SKItem_{item}.pktagreseller \rightarrow$$
$$send.Msg3.T.S.Encrypt.SK_T.PKItem(item).pktagreseller \rightarrow SKIP$$

A renaming is applied to the definitions of the reseller, supplier and the TGC to allow the adversary to intercept and fake messages on the communication channels. The input can come either from the channel $receive$ or from the channel $fake$ and the output can go to either $send$ or $intercept$. The supplier, reseller, and TGC cannot tell whether a message has been sent from the other party or from the adversary.

$$RESELLER_0 \mathrel{\widehat{=}} RESELLER(item, pktagreseller, PK_T)$$
$$[[send.Msg1 \leftarrow send.Msg1, send.Msg1 \leftarrow intercept.Msg1,$$
$$receive.Msg4 \leftarrow receive.Msg4, receive.Msg4 \leftarrow fake.Msg4]]$$

$$SUPPLIER_0 \mathrel{\widehat{=}} SUPPLIER(item, SKItem_{item}, PKItem_{item}, PK_T)$$
$$[[receive.Msg1 \leftarrow receive.Msg1, receive.Msg1 \leftarrow fake.Msg1,$$
$$send.Msg2 \leftarrow send.Msg2, send.Msg2 \leftarrow intercept.Msg2,$$
$$receive.Msg3 \leftarrow receive.Msg3, receive.Msg3 \leftarrow fake.Msg3,$$
$$send.Msg4 \leftarrow send.Msg4, send.Msg4 \leftarrow intercept.Msg4]]$$

$$TGC_0 \mathrel{\widehat{=}} TGC(SK_T, PK_T)$$
$$[[receive.Msg2 \leftarrow receive.Msg2, receive.Msg2 \leftarrow fake.Msg2,$$

$send.Msg3 \leftarrow send.Msg3, send.Msg3 \leftarrow intercept.Msg3]]$

The Adversary

The adversary is parametrised by the information it knows. In this process the adversary is parametrised by $m1s$ which is the set of all message 1s the adversary knows, $m2s$ which is the set of all message 2s the adversary knows, $m3s$ which is the set of all message 3s the adversary knows, $m4s$ which is the set of all message 4s the adversary knows, and tk which is the set of all $Tags$ the adversary knows. If the adversary knows the key to decrypt a message it learns the contents of the message, otherwise the adversary learns the encrypted content of the message. The adversary can fake new messages using any values it has learnt, it can also fake messages by replaying the encrypted content of a message it has seen. The value $KnownKeys = \{SKItem_{item}, SK_S, SK_T, SK_A, PK_A\}$ is the set of all keys the adversary knows. The CSP definition is:

$I(m1s, m2s, m3s, m4s, tk) \mathrel{\widehat{=}}$
$receive.Msg1?a.b.tk' \rightarrow$
$I(m1s, m2s, m3s, m4s, tk \cup \{tk'\})$
$\Box\ intercept.Msg1?a.b.tk' \rightarrow$
$I(m1s, m2s, m3s, m4s, tk \cup \{tk'\})$
$\Box\ receive.Msg2?b.a.Encrypt.k.tk' \rightarrow$
if $k \in KnownKeys$ then $I(m1s, m2s, m3s, m4s, tk \cup \{tk'\})$
else $I(m1s, m2s \cup \{Encrypt.k.tk'\}, m3s, m4s, tk)$
$\Box\ intercept.Msg2?b.a.Encrypt.k.tk' \rightarrow$
if $k \in KnownKeys$ then $I(m1s, m2s, m3s, m4s, tk \cup \{tk'\})$
else $I(m1s, m2s \cup \{Encrypt.k.tk'\}, m3s, m4s, tk)$
$\Box\ receive.Msg3?b.a.Encrypt.k.pk.tk' \rightarrow$
if $k \in KnownKeys$ then $I(m1s, m2s, m3s, m4s, tk \cup \{tk'\})$
else $I(m1s, m2s, m3s \cup \{Encrypt.k.pk.tk'\}, m4s, tk)$
$\Box\ intercept.Msg3?b.a.Encrypt.k.pk.tk' \rightarrow$
if $k \in KnownKeys$ then $I(m1s, m2s, m3s, m4s, tk \cup \{tk'\})$
else $I(m1s, m2s, m3s \cup \{Encrypt.k.pk.tk'\}, m4s, tk)$
$\Box\ receive.Msg4?b.a.Encrypt.k.pk.tk' \rightarrow$
if $k \in KnownKeys$ then $I(m1s, m2s, m3s, m4s, tk \cup \{tk'\})$

$$
\begin{aligned}
&\quad \text{else } I(m1s, m2s, m3s, m4s \cup \{Encrypt.k.pk.tk'\}, tk) \\
&\Box\ intercept.Msg4?b.a.Encrypt.k.pk.tk' \rightarrow \\
&\quad \text{if } k \in KnownKeys \text{ then } I(m1s, m2s, m3s, m4s, tk \cup \{tk'\}) \\
&\quad \text{else } I(m1s, m2s, m3s, m4s \cup \{Encrypt.k.pk.tk'\}, tk) \\
&fake.Msg1?a.b?m : m1s \rightarrow I(m1s, m2s, m3s, m4s, tk) \\
&fake.Msg2?b.a?m : m2s \rightarrow I(m1s, m2s, m3s, m4s, tk) \\
&fake.Msg3?b.a?m : m3s \rightarrow I(m1s, m2s, m3s, m4s, tk) \\
&fake.Msg4?b.a?m : m4s \rightarrow I(m1s, m2s, m3s, m4s, tk) \\
&fake.Msg1?a.b?tk' : tk \rightarrow I(m1s, m2s, m3s, m4s, tk) \\
&fake.Msg2?a.b!Encrypt?k?tk' : tk \rightarrow I(m1s, m2s, m3s, m4s, tk) \\
&fake.Msg3?a.b!Encrypt?k?pk?tk' : tk \rightarrow I(m1s, m2s, m3s, m4s, tk) \\
&fake.Msg4?a.b!Encrypt?k?pk?tk' : tk \rightarrow I(m1s, m2s, m3s, m4s, tk)
\end{aligned}
$$

In this model the adversary has the initial knowledge of $pktagwrong$:

$$INTRUDER \mathrel{\widehat{=}} I(\{\}, \{\}, \{\}, \{\}, \{pktagwrong\})$$

Verification

The model of the protocol is now put together with the intruder:

$$CHANS \mathrel{\widehat{=}} \{|send, receive|\}$$

$$AGENTS \mathrel{\widehat{=}} RESELLER_0 |[CHANS]| SUPPLIER_0 |[CHANS]| TGC_0$$

$$SYSTEM \mathrel{\widehat{=}} AGENTS |[\{|send, receive, fake, intercept|\}]| INTRUDER$$

Figure A.4 illustrates the processes and channels in the model of the protocol. The supplier, reseller, and the TGC processes have access to the channels $send$ and $receive$. The supplier also has access to the channel $running$ and the reseller has access to the channel $commit$. The adversary has access to the channels $fake$ and $intercept$.

FDR takes as input an implementation of the protocol and a specification and checks that the implementation refines the specification. The generation of a tag has to resist a fabrication attack. Informally, if the reseller receives a tag that it accepts, the supplier must have sent a message to the TGC requesting the tag. In CSP the specification is defined as:

$$SPEC_0 \mathrel{\widehat{=}} running.S.R.pktagcustomer \rightarrow commit.R?S.pktagcustomer \rightarrow SPEC$$

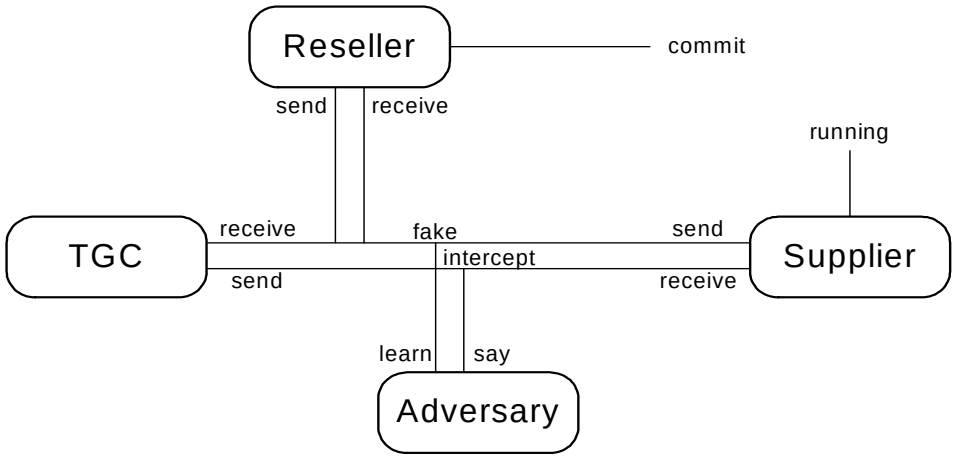

Figure A.2: CSP Network Model: Supplier Generating a Tag

$$SPEC_1 \mathrel{\widehat{=}} \{|running.S.R.pktagcustomer, commit.R.S.pktagcustomer|\}$$

$$SPEC \mathrel{\widehat{=}} SPEC_0 \,|||\, RUN(\Sigma \setminus SPEC_1)$$

The term Σ is the set of all possible events. The FDR model checker checks $SPEC \sqsubseteq_{\mathrm{T}} SYSTEM$ and returns TRUE after 111 states with 387 transitions.

A.1.3 Reseller Generating Tag

In this phase the reseller generates a tag for a customer with the TGC.

Sets

$$Agents = \{C, T, R, A\}$$

$$Items = \{item\}$$

$$Tags = \{pktag, pktag2, pktagwrong, pktagwrong2\}$$

$$ActualTags = \{tag, tag2, tagwrong, tagwrong2\}$$

$$ItemKeys = \{PKItem_{item}, SKItem_{item}\}$$

$$Key = \{PK_R, SK_R, PK_T, SK_T, PK_C, SK_C, PK_A, SK_A\}$$

The set $Agents$ is composed of the participants involved when the reseller generates the tag with the TGC and the processes in the CSP model. The values C, T, R, and A represent the customer, TGC, reseller, and adversary respectively. The set $Items$ is composed of the items that the customer wants to purchase.

The set $Tags$ is composed of the one time keys in the tags. The set $ItemKeys$ is composed of the private and public keys that are associated with the item where $PKItem$ is the public key and $SKItem$ is the private key. The set Key is the set of public and private keys associated with each Agent where PK is the public key and SK is the private key.

Messages

The three messages that are used when the reseller generates a tag in the tagged transaction protocol are denoted $MSG1$, $MSG2$, and $MSG3$:

$$MSG1 \mathrel{\widehat{=}} \{Msg1.a.b.pktag.pktag2 \mid a \in Agents, b \in Agents, pktag \in Tags, pktag2 \in Tags\}$$

$$MSG2 \mathrel{\widehat{=}} \{Msg2.a.b.Encrypt.k.tag.pktag.tag2.pktag2 \mid a \in Agents, b \in Agents, k \in Keys, tag \in ActualTags, pktag \in Tags, tag2 \in ActualTags, pktag2 \in Tags\}$$

$$MSG3 \mathrel{\widehat{=}} \{Msg3.a.b.Encrypt.k.itemkey.pktag.Encrypt.k.itemkey.pktag2 \mid a \in Agents, b \in Agents, k \in Key, itemkey \in ItemKeys, pktag \in Tags, pktag2 \in Tags\}$$

The set of all messages in this phase of the protocol is defined as:

$$MSG \mathrel{\widehat{=}} MSG1 \cup MSG2 \cup MSG3$$

The channels defined in this system are $send$, $receive$, $intercept$, $fake$, $running$, $running2$, $commit$, and $commit2$. The channels $send$ and $receive$ are used by the processes to exchange messages. The channels $intercept$, and $fake$ are used by the adversary to intercept messages and insert new messages. The channels $running$, $running2$, $commit$ and $commit2$ are used to represent the state of the processes. Formally in CSP the channels are defined as:
channel $send, receive, intercept, fake : MSG$
channel $running, commit$
channel $running2, commit2$

Processes

The customer is defined as:

$$CUSTOMER(item, pktag, pktag2, PK_T) \mathrel{\widehat{=}}$$
$$send.Msg1.C.R.pktag.pktag2 \rightarrow$$
$$receive.Msg3.T.C.Encrypt.SK_T.PKItem_{item}.pktag.$$
$$Encrypt.SK_T.PKItem_{item}.pktag2 \rightarrow$$
$$commit.C.T.pktag.pktag2 \rightarrow$$
$$commit2.C.R.pktag.pktag2 \rightarrow SKIP$$

the reseller is defined as:

$$RESELLER(item, tag, tag2, SK_R, PK_R, PK_T) \mathrel{\widehat{=}}$$
$$receive.Msg1.C.R.pktag.pktag2 \rightarrow$$
$$send.Msg2.R.T.Encrypt.SK_R.tag.pktag.tag2.pktag2 \rightarrow$$
$$running2.R.C.pktag.pktag2 \rightarrow SKIP$$

and the TGC is defined as:

$$TGC(SK_T, PK_T, PK_R) \mathrel{\widehat{=}}$$
$$receive.Msg2.R.T.Encrypt.SK_R.tag.pktag.tag2.pktag2 \rightarrow$$
$$running.T.C.pktag.pktag2 \rightarrow$$
$$send.Msg3.T.C.Encrypt.SK_T.PKItem_{item}.pktag.$$
$$Encrypt.SK_T.PKItem_{item}.pktag2 \rightarrow SKIP$$

A renaming is applied to the definitions of the customer, reseller, and the TGC to allow the adversary to intercept and fake messages on the communication channels. The input can come either from the channel $receive$ or from the channel $fake$ and the output can go to either $send$ or $intercept$. The customer, reseller, and TGC cannot tell whether a message has been sent from the other party or from the adversary.

$$CUSTOMER_0 \mathrel{\widehat{=}} CUSTOMER(item, pktag, pktag2, PK_T)$$
$$[[send.Msg1 \leftarrow send.Msg1, send.Msg1 \leftarrow intercept.Msg1,$$
$$receive.Msg3 \leftarrow receive.Msg3, receive.Msg3 \leftarrow fake.Msg3]]$$

$$RESELLER_0 \mathrel{\widehat{=}} RESELLER(item, tag, tag2, SK_R, PK_R, PK_T)$$
$$[[receive.Msg1 \leftarrow receive.Msg1, receive.Msg1 \leftarrow fake.Msg1,$$
$$send.Msg2 \leftarrow send.Msg2, send.Msg2 \leftarrow intercept.Msg2]]$$

$TGC_0 \mathrel{\widehat{=}} TGC(SK_T, PK_T, PK_R)$
$[[receive.Msg2 \leftarrow receive.Msg2, receive.Msg2 \leftarrow fake.Msg2,$
$send.Msg3 \leftarrow send.Msg3, send.Msg3 \leftarrow intercept.Msg3]]$

The Adversary

The adversary is parametrised by the information it knows. In this process the adversary is parametrised by $m1s$ which is the set of all message 1s the adversary knows, $m2s$ which is the set of all message 2s the adversary knows, $m3s$ which is the set of all message 3s the adversary knows, tk which is the set of all $Tags$ the adversary knows, and at which is the set of all $ActualTags$ the adversary knows. If the adversary knows the key to decrypt a message it learns the contents of the message, otherwise the adversary learns the encrypted content of the message. The adversary can fake new messages using any values it has learnt, it can also fake messages by replaying the encrypted content of a message it has seen. The value $KnownKeys = \{SKItem_{item}, SK_C, SK_R, SK_T, SK_A, PK_A\}$ is the set of all keys known by the adversary. The CSP definition is:

$I(m1s, m2s, m3s, tk, at) \mathrel{\widehat{=}}$
$receive.Msg1?a.b.tk'.tk2' \rightarrow$
$I(m1s, m2s, m3s, tk \cup \{tk', tk2'\}, at)$
$\Box\ intercept.Msg1?a.b.tk'.tk2' \rightarrow$
$I(m1s, m2s, m3s, tk \cup \{tk', tk2'\}, at)$
$\Box\ receive.Msg2?b.a.Encrypt.k.at'.tk'.at2'.tk2' \rightarrow$
if $k \in KnownKeys$ then $I(m1s, m2s, m3s, tk \cup \{tk', tk2'\},$
$at \cup \{at', at2'\})$
else $I(m1s, m2s \cup \{Encrypt.k.at'.tk'.at2'.tk2'\}, m3s, tk, at)$
$\Box\ intercept.Msg2?b.a.Encrypt.k.at'.tk'.at2'.tk2' \rightarrow$
if $k \in KnownKeys$ then $I(m1s, m2s, m3s, tk \cup \{tk', tk2'\},$
$at \cup \{at', at2'\})$
else $I(m1s, m2s \cup \{Encrypt.k.at'.tk'.at2'.tk2'\}, m3s, tk, at)$
$\Box\ receive.Msg3?b.a.Encrypt.k.pk.tk'.Encrypt.k.pk.tk2' \rightarrow$
if $k \in KnownKeys$ then $I(m1s, m2s, m3s, tk \cup \{tk', tk2'\}, at)$
else $I(m1s, m2s, m3s \cup \{Encrypt.k.pk.tk'.Encrypt.k.pk.tk2'\},$

$$
\begin{aligned}
&\qquad tk, at) \\
&\Box\ intercept.Msg3?b.a.Encrypt.k.pk.tk'.Encrypt.k.pk.tk2' \rightarrow \\
&\qquad \text{if } k \in KnownKeys \text{ then } I(m1s, m2s, m3s, tk \cup \{tk', tk2'\}, at) \\
&\qquad \text{else } I(m1s, m2s, m3s \cup \{Encrypt.k.pk.tk'.Encrypt.k.pk.tk2'\}, \\
&\qquad\qquad tk, at) \\
&fake.Msg1?a.b?m : m1s \rightarrow I(m1s, m2s, m3s, tk, at) \\
&fake.Msg2?b.a?m : m2s \rightarrow I(m1s, m2s, m3s, tk, at) \\
&fake.Msg3?b.a?m : m3s \rightarrow I(m1s, m2s, m3s, tk, at) \\
&fake.Msg1?a.b?tk' : tk?tk2' : tk \rightarrow I(m1s, m2s, m3s, tk, at) \\
&fake.Msg2?a, b!Encrypt?k?at' : at?tk' : tk?at2' : at?tk2' : tk \rightarrow \\
&\qquad\qquad I(m1s, m2s, m3s, tk, at) \\
&fake.Msg3?a.b!Encrypt?k?pk?tk' : tk!Encrypt?k?pk?tk2' : tk \rightarrow \\
&\qquad\qquad I(m1s, m2s, m3s, tk, at)
\end{aligned}
$$

This model considers an adversary with the initial knowledge of $pktagwrong$, $pktagwrong2$, $tagwrong$, and $tagwrong2$:

$$INTRUDER \mathrel{\widehat{=}} I(\{\}, \{\}, \{\}, \{pktagwrong, pktagwrong2\}, \{tagwrong, tagwrong2\})$$

Verification

The model of the protocol is now put together with the intruder:

$$CHANS \mathrel{\widehat{=}} \{|send, receive|\}$$

$$AGENTS \mathrel{\widehat{=}} CUSTOMER_0 |[CHANS]| RESELLER_0 |[CHANS]| TGC_0$$

$$SYSTEM \mathrel{\widehat{=}} AGENTS |[\{|send, receive, fake, intercept|\}]| INTRUDER$$

Figure A.5 illustrates the processes and channels in the model of the protocol. The customer, reseller, and the TGC processes have access to the channels $send$ and $receive$. The TGC has access to the channel $running$, the reseller has access to the channel $running2$ and the customer has access to the channels $commit$ and $commit2$. The adversary has access to the channels $fake$ and $intercept$.

FDR takes as input an implementation of the protocol and a specification and checks that the implementation refines the specification. This model has two specifications for the reseller generating the tag. The first specification prevents replay. Informally, if the customer receives a tag then two different values must have been signed by the TGC. In CSP the specification is defined as:

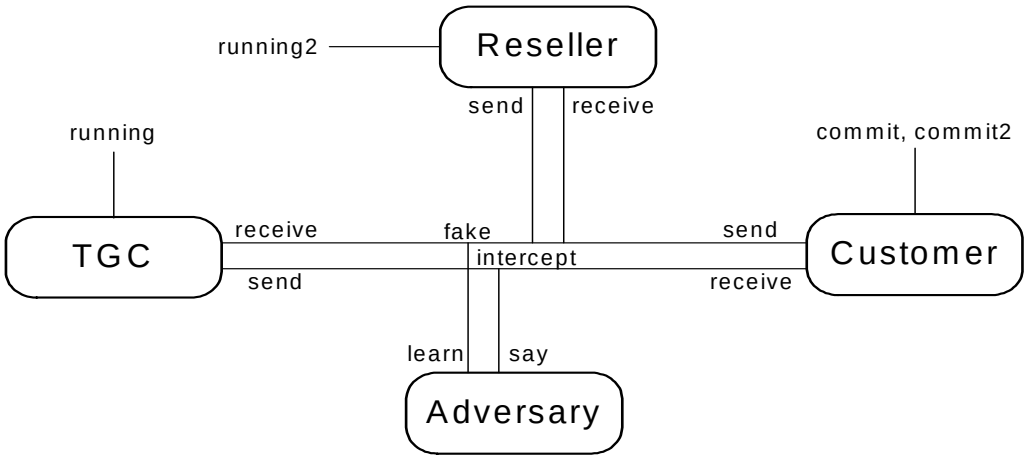

Figure A.3: CSP Network Model: Reseller Generating a Tag

$SPEC_0 \mathrel{\widehat{=}}$

$running.T.C.pktagcustomer.pktagcustomer2 \rightarrow$
$commit.C?T.pktagcustomer.pktagcustomer2 \rightarrow SPEC$

$SPEC_1 \mathrel{\widehat{=}}$

$\{|running.T.C.pktagcustomer.pktagcustomer2,$
$commit.C.T.pktagcustomer.pktagcustomer2|\}$

$$SPEC \mathrel{\widehat{=}} SPEC_0 \,|||\, RUN(\Sigma \setminus SPEC_1)$$

The term Σ is the set of all possible events. The FDR model checker checks $SPEC \sqsubseteq_{\mathrm{T}} SYSTEM$ and returns TRUE after 457 states with 2005 transitions.

The second specification checks for a network sniffing attack. Informally, if a customer receives tags signed by the TGC then the reseller must have sent a message to the TGC to request the generation of the tags. In CSP the specification is defined as:

$SPEC2_0 \mathrel{\widehat{=}}$

$running2.R.C.pktagcustomer.pktagcustomer2 \rightarrow$
$commit2.C?R.pktagcustomer.pktagcustomer2 \rightarrow SPEC$

$SPEC2_1 \mathrel{\widehat{=}}$

$\{|running2.R.C.pktagcustomer.pktagcustomer2,$
$commit2.C.R.pktagcustomer.pktagcustomer2|\}$

$$SPEC2 \mathrel{\widehat{=}} SPEC2_0 \,|||\, RUN(\Sigma \setminus SPEC2_1)$$

The term Σ is the set of all possible events. The FDR model checker checks $SPEC2 \sqsubseteq_{\mathrm{T}} SYSTEM$ and returns TRUE after 1423 states with 4835 transitions.

A.2 Extended Tagged Transaction Protocol

The extended tagged transaction protocol has two models: the supplier generating the tag, and a reseller generating a tag. The registration model is not repeated as it is the same as in the tagged transaction protocol.

A.2.1 Supplier Generating Tag

In this phase the supplier is generating a tag for a reseller with the TGC.

Sets

$$Agents = \{S, T, R, A\}$$

$$Items = \{item\}$$

$$Tags = \{pktagreseller, pktagwrong\}$$

$$ItemKeys = \{PKItem_{item}, SKItem_{item}\}$$

$$Token = tokenreseller$$

$$Key = \{PK_S, SK_S, PK_T, SK_T, PK_C, SK_C, PK_A, SK_A\}$$

The set $Agents$ is composed of the participants involved when the supplier generates the tag with the TGC and the processes in the CSP model. The values S, T, R, and A represent the supplier, TGC, reseller, and adversary respectively. The set $Items$ is composed of the items that the reseller wants to purchase. The set $Tags$ is composed of the one time keys in the tags. The set $ItemKeys$ is composed of the private and public keys that are associated with the item where $PKItem$ is the public key and $SKItem$ is the private key. The set Key is the set of public and private keys associated with each Agent where PK is the public key and SK is the private key.

Messages

The four messages that are part of the registration phase of the tagged transaction protocol are denoted $MSG1$, $MSG2$, $MSG3$, and $MSG4$:

$$MSG1 \mathrel{\widehat{=}} \{Msg1.a.b.token.tag \mid a \in Agents, b \in Agents, token \in Token, tag \in Tags\}$$

$$MSG2 \mathrel{\widehat{=}} \{Msg2.a.b.Encrypt.k.token.tag \mid a \in Agents, b \in Agents, k \in ItemKeys, token \in Token, tag \in Tags\}$$

$$MSG3 \mathrel{\widehat{=}} \{Msg3.a.b.Encrypt.k.itemkey.token.tag \mid a \in Agents, b \in Agents, k \in Key, itemkey \in ItemKeys, token \in Token, tag \in Tags\}$$

$$MSG4 \mathrel{\widehat{=}} \{Msg4.a.b.Encrypt.k.itemkey.token.tag \mid a \in Agents, b \in Agents, k \in Key, itemkey \in ItemKeys, token \in Token, tag \in Tags\}$$

The set of all messages in this phase of the protocol is defined as:

$$MSG \mathrel{\widehat{=}} MSG1 \cup MSG2 \cup MSG3 \cup MSG4$$

The channels defined in this system are $send$, $receive$, $intercept$, $fake$, $running$, and $commit$. The channels $send$ and $receive$ are used by the processes to exchange messages. The channels $intercept$, and $fake$ are used by the adversary to intercept messages and insert new messages. The channels $running$ and $commit$ are used to represent the state of the processes. Formally in CSP the channels are defined as:
channel $send, receive, intercept, fake : MSG$
channel $running, commit$

Processes

The reseller is defined as:

$$RESELLER(item, tokenreseller, pktagreseller, PK_T) \mathrel{\widehat{=}}$$
$$send.Msg1.R.S.tokenreseller.pktagreseller \rightarrow$$
$$receive.Msg4.S.R.Encrypt.SK_T.PKItem_{item}.tokenreseller.pktagreseller \rightarrow$$
$$commit.R.S.tokenreseller.pktagcustomer \rightarrow SKIP$$

The supplier is defined as:

$$SUPPLIER(item, SKItem_{item}, PKItem_{item}, PK_T) \mathrel{\widehat{=}}$$
$$receive.Msg1.R.S.tokenreseller.pktagreseller \rightarrow$$
$$running.R.S.tokenreseller.pktagreseller \rightarrow$$
$$send.Msg2.S.T.Encrypt.SKItem_{item}.tokenreseller.pktagreseller \rightarrow$$
$$receive.Msg3.T.S.Encrypt.SK_T.PKItem(item).tokenreseller.$$
$$pktagreseller \rightarrow$$
$$send.Msg4.S.R.Encrypt.SK_T.PKItem(item).tokenreseller.$$
$$pktagreseller \rightarrow SKIP$$

and the TGC is defined as:

$$TGC(SK_T, PK_T) \mathrel{\widehat{=}}$$
$$receive.Msg2.S.T.Encrypt.SKItem_{item}.tokenreseller.pktagreseller \rightarrow$$
$$send.Msg3.T.S.Encrypt.SK_T.PKItem(item).tokenreseller.$$
$$pktagreseller \rightarrow SKIP$$

A renaming is applied to the definitions of the reseller, supplier and the TGC to allow the adversary to intercept and fake messages on the communication channels. The input can come either from the channel $receive$ or from the channel $fake$ and the output can go to either $send$ or $intercept$. The supplier, reseller, and TGC cannot tell whether a message has been sent from the other party or from the adversary.

$$RESELLER_0 \mathrel{\widehat{=}} RESELLER(item, pktagreseller, PK_T)$$
$$[[send.Msg1 \leftarrow send.Msg1, send.Msg1 \leftarrow intercept.Msg1,$$
$$receive.Msg4 \leftarrow receive.Msg4, receive.Msg4 \leftarrow fake.Msg4]]$$

$$SUPPLIER_0 \mathrel{\widehat{=}} SUPPLIER(item, SKItem_{item}, PKItem_{item}, PK_T)$$
$$[[receive.Msg1 \leftarrow receive.Msg1, receive.Msg1 \leftarrow fake.Msg1,$$

$send.Msg2 \leftarrow send.Msg2, send.Msg2 \leftarrow intercept.Msg2,$
$receive.Msg3 \leftarrow receive.Msg3, receive.Msg3 \leftarrow fake.Msg3,$
$send.Msg4 \leftarrow send.Msg4, send.Msg4 \leftarrow intercept.Msg4]]$

$TGC_0 \mathrel{\widehat{=}} TGC(SK_T, PK_T)$
$[[receive.Msg2 \leftarrow receive.Msg2, receive.Msg2 \leftarrow fake.Msg2,$
$send.Msg3 \leftarrow send.Msg3, send.Msg3 \leftarrow intercept.Msg3]]$

The Adversary

The adversary is parametrised by the information it knows. In this process the adversary is parametrised by $m1s$ which is the set of all message 1s the adversary knows, $m2s$ which is the set of all message 2s the adversary knows, $m3s$ which is the set of all message 3s the adversary knows, $m4s$ which is the set of all message 4s the adversary knows, tg which is the set of all $Tags$ the adversary knows, and tk which is the set of all $Token$ the adversary knows. If the adversary knows the key to decrypt a message it learns the contents of the message, otherwise the adversary learns the encrypted content of the message. The adversary can fake new messages using any values it has learnt, it can also fake messages by replaying the encrypted content of a message it has seen. The value $KnownKeys = \{SKItem_{item}, SK_S, SK_T, SK_A, PK_A\}$is defined as the keys the adversary knows. The CSP definition is:

$I(m1s, m2s, m3s, m4s, tk, tg) \mathrel{\widehat{=}}$
$receive.Msg1?a.b.tk'.tg' \rightarrow$
$I(m1s, m2s, m3s, m4s, tk \cup \{tk'\}, tg \cup \{tg'\})$
$\Box\ intercept.Msg1?a.b.tk'.tg' \rightarrow$
$I(m1s, m2s, m3s, m4s, tk \cup \{tk'\}, tg \cup \{tg'\})$
$\Box\ receive.Msg2?b.a.Encrypt.k.tk'.tg' \rightarrow$
if $k \in KnownKeys$ then $I(m1s, m2s, m3s, m4s, tk \cup \{tk'\},$
$tg \cup \{tg'\})$
else $I(m1s, m2s \cup \{Encrypt.k.tk'.tg'\}, m3s, m4s, tk, tg)$
$\Box\ intercept.Msg2?b.a.Encrypt.k.tk'.tg' \rightarrow$
if $k \in KnownKeys$ then $I(m1s, m2s, m3s, m4s, tk \cup \{tk'\},$
$tg \cup \{tg'\})$

$$
\begin{aligned}
&\text{else } I(m1s, m2s \cup \{Encrypt.k.tk'.tg'\}, m3s, m4s, tk, tg)\\
&\Box\ receive.Msg3?b.a.Encrypt.k.pk.tk'.tg' \rightarrow\\
&\quad \text{if } k \in KnownKeys \text{ then } I(m1s, m2s, m3s, m4s, tk \cup \{tk'\},\\
&\qquad\qquad tg \cup \{tg'\})\\
&\quad \text{else } I(m1s, m2s, m3s \cup \{Encrypt.k.pk.tk'.tg'\}, m4s, tk, tg)\\
&\Box\ intercept.Msg3?b.a.Encrypt.k.pk.tk'.tg' \rightarrow\\
&\quad \text{if } k \in KnownKeys \text{ then } I(m1s, m2s, m3s, m4s, tk \cup \{tk'\},\\
&\qquad\qquad tg \cup \{tg'\})\\
&\quad \text{else } I(m1s, m2s, m3s \cup \{Encrypt.k.pk.tk'.tg'\}, m4s, tk, tg)\\
&\Box\ receive.Msg4?b.a.Encrypt.k.pk.tk'.tg' \rightarrow\\
&\quad \text{if } k \in KnownKeys \text{ then } I(m1s, m2s, m3s, m4s, tk \cup \{tk'\},\\
&\qquad\qquad tg \cup \{tg'\})\\
&\quad \text{else } I(m1s, m2s, m3s, m4s \cup \{Encrypt.k.pk.tk'.tg'\}, tk, tg)\\
&\Box\ intercept.Msg4?b.a.Encrypt.k.pk.tk'.tg' \rightarrow\\
&\quad \text{if } k \in KnownKeys \text{ then } I(m1s, m2s, m3s, m4s, tk \cup \{tk'\},\\
&\qquad\qquad tg \cup \{tg'\})\\
&\quad \text{else } I(m1s, m2s, m3s, m4s \cup \{Encrypt.k.pk.tk'.tg'\}, tk, tg)\\
&fake.Msg1?a.b?m : m1s \rightarrow I(m1s, m2s, m3s, m4s, tk, tg)\\
&fake.Msg2?b.a?m : m2s \rightarrow I(m1s, m2s, m3s, m4s, tk, tg)\\
&fake.Msg3?b.a?m : m3s \rightarrow I(m1s, m2s, m3s, m4s, tk, tg)\\
&fake.Msg4?b.a?m : m4s \rightarrow I(m1s, m2s, m3s, m4s, tk, tg)\\
&fake.Msg1?a.b?tk' : tk?tg' : tg \rightarrow I(m1s, m2s, m3s, m4s, tk, tg)\\
&fake.Msg2?a.b!Encrypt?k?tk' : tk?tg' : tg \rightarrow\\
&\quad I(m1s, m2s, m3s, m4s, tk, tg)\\
&fake.Msg3?a.b!Encrypt?k?pk?tk' : tk?tg' : tg \rightarrow\\
&\quad I(m1s, m2s, m3s, m4s, tk, tg)\\
&fake.Msg4?a.b!Encrypt?k?pk?tk' : tk?tg' : tg \rightarrow\\
&\quad I(m1s, m2s, m3s, m4s, tk, tg)
\end{aligned}
$$

In this model the adversary has the initial knowledge of $pktagwrong$ and $tokenreseller$:

$$INTRUDER \mathrel{\widehat{=}} I(\{\}, \{\}, \{\}, \{\}, \{tokenreseller\}, \{pktagwrong\})$$

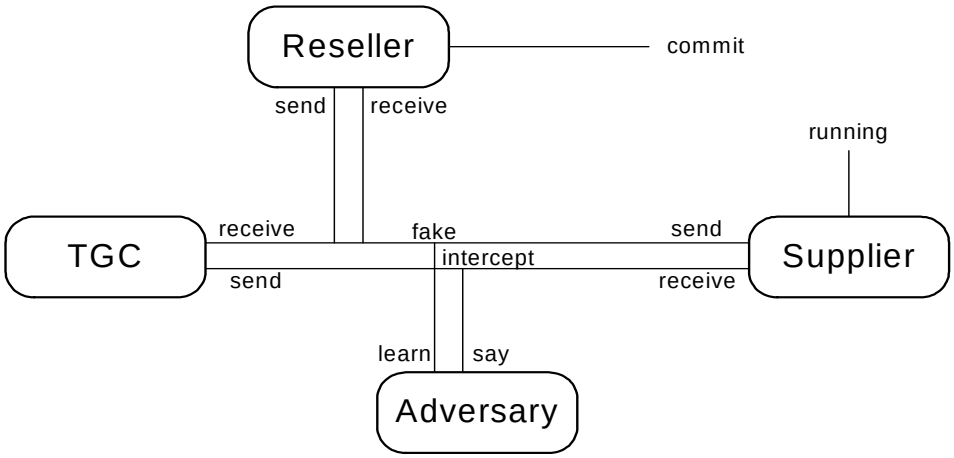

Figure A.4: CSP Network Model: Supplier Generating a Tag

Verification

The model of the protocol is now put together with the intruder:

$$CHANS \mathrel{\widehat{=}} \{|send, receive|\}$$

$$AGENTS \mathrel{\widehat{=}} RESELLER_0 |[CHANS]| SUPPLIER_0 |[CHANS]| TGC_0$$

$$SYSTEM \mathrel{\widehat{=}} AGENTS |[\{|send, receive, fake, intercept|\}]| INTRUDER$$

Figure A.4 illustrates the processes and channels in the model of the protocol. The supplier, reseller, and the TGC processes have access to the channels $send$ and $receive$. The supplier also has access to the channel $running$ and the reseller has access to the channel $commit$. The adversary has access to the channels $fake$ and $intercept$.

FDR takes as input an implementation of the protocol and a specification and checks that the implementation refines the specification. The generation of a tag has to resist a fabrication attack. Informally, if the reseller receives a tag that it accepts, the supplier must have sent a message to the TGC requesting the tag. In CSP the specification is defined as:

$SPEC_0 \mathrel{\widehat{=}}$

$running.S.R.tokenreseller.pktagcustomer \rightarrow$

$commit.R?S.tokenreseller.pktagcustomer \rightarrow SPEC$

$SPEC_1 \mathrel{\widehat{=}}$

$\{|running.S.R.pktagcustomer.tokenreseller,$
$commit.R.S.pktagcustomer.tokenreseller|\}$

$$SPEC \mathrel{\widehat{=}} SPEC_0 \,|||\, RUN(\Sigma \setminus SPEC_1)$$

The term Σ is the set of all possible events. The FDR model checker checks $SPEC \sqsubseteq_{\mathrm{T}} SYSTEM$ and returns TRUE after 111 states with 387 transitions.

A.2.2 Reseller Generating Tag

In this phase, the reseller is generating a tag for the customer with the TGC.

Sets

$$Agents = \{C, T, R, A\}$$

$$Items = \{item\}$$

$$Tags = \{pktag, pktag2, pktagwrong, pktagwrong2\}$$

$$ActualTags = \{tag, tag2, tagwrong, tagwrong2\}$$

$$Token = \{tktag, tktag2\}$$

$$Challenge = \{c1, c2\}$$

$$Response = \{r1, r2\}$$

$$ItemKeys = \{PKItem_{item}, SKItem_{item}\}$$

$$Key = \{PK_R, SK_R, PK_T, SK_T, PK_C, SK_C, PK_A, SK_A\}$$

The set $Agents$ is composed of the participants involved when the reseller generates the tag with the TGC and the processes in the CSP model. The values C, T, R, and A represent the customer, TGC, reseller, and adversary respectively. The set $Items$ is composed of the items that the customer wants to purchase. The set $Tags$ is composed of the one time keys in the tags. The set $ActualTags$ is composed of the tags that have previously been generated and sent to the reseller. The set $Token$ represents the identity tokens. The sets $Challenge$ and $Response$ represent the challenge and response values used in the protocol. The set $ItemKeys$ is composed of the private and public keys that are associated with

the item where $PKItem$ is the public key and $SKItem$ is the private key. The set Key is the set of public and private keys associated with each Agent where PK is the public key and SK is the private key.

Messages

The five messages that are used when the reseller generates a tag in the extended tagged transaction protocol are denoted $MSG1$, $MSG2$, $MSG3$, $MSG4$, and $MSG5$:

$$\begin{aligned}MSG1 \mathrel{\widehat{=}} \{&Msg1.a.b.token.pktag.token2.pktag2 \mid \\ &a \in Agents, b \in Agents, token \in Token, pktag \in Tags, \\ &token2 \in Token, pktag2 \in Tags\}\end{aligned}$$

$$\begin{aligned}MSG2 \mathrel{\widehat{=}} \{&Msg2.a.b.Encrypt.k.tag.token.pktag.tag2.token2.pktag2 \mid \\ &a \in Agents, b \in Agents, k \in Keys, tag \in ActualTags, \\ &token \in Token, pktag \in Tags, \\ &tag2 \in ActualTags, token2 \in Token, pktag2 \in Tags\}\end{aligned}$$

$$\begin{aligned}MSG3 \mathrel{\widehat{=}} \{&Msg3.a.b.c1.c2\} \mid \\ &a \in Agents, b \in Agents, c1 \in Challenge, c2 \in Challenge\end{aligned}$$

$$\begin{aligned}MSG4 \mathrel{\widehat{=}} \{&Msg4.a.b.Encrypt.k.c1.c2.r1.r2\} \mid \\ &a \in Agents, b \in Agents, c1 \in Challenge, c2 \in Challenge, \\ &r1 \in Response, r2 \in Response\end{aligned}$$

$$\begin{aligned}MSG5 \mathrel{\widehat{=}} \{&Msg5.a.b.Encrypt.k.itemkey.token.pktag.Encrypt.k.itemkey. \\ &\qquad token2.pktag2 \mid \\ &a \in Agents, b \in Agents, k \in Key, itemkey \in ItemKeys, \\ &token \in Token, pktag \in Tags, token2 \in Token, pktag2 \in Tags\}\end{aligned}$$

The set of all messages in this phase of the protocol is defined as:

$$MSG \mathrel{\widehat{=}} MSG1 \cup MSG2 \cup MSG3 \cup MSG4 \cup MSG5$$

The channels defined in this system are $send$, $receive$, $intercept$, $fake$, $running$, $running2$, $commit$, and $commit2$. The channels $send$ and $receive$ are used by the

processes to exchange messages. The channels $intercept$, and $fake$ are used by the adversary to intercept messages and insert new messages. The channels $running$, $running2$, $commit$ and $commit2$ are used to represent the state of the processes. Formally in CSP the channels are defined as:
channel $send, receive, intercept, fake : MSG$
channel $running, commit$
channel $running2, commit2$

Processes

The customer is defined as:

$$CUSTOMER(item, tktag, pktag, tktag2, pktag2, PK_T) \mathrel{\widehat{=}}$$
$$send.Msg1.C.R.tktag.pktag.tktag2.pktag2 \rightarrow$$
$$receive.Msg5.T.C.Encrypt.SK_T.PKItem_{item}.tktag.pktag.$$
$$\quad Encrypt.SK_T.PKItem_{item}.tktag2.pktag2 \rightarrow$$
$$commit.C.T.tktag.pktag.tktag2.pktag2 \rightarrow$$
$$commit2.C.R.tktag.pktag.tktag2.pktag2 \rightarrow SKIP$$

the reseller is defined as:

$$RESELLER(item, tag, tag2, r1, r2, SK_R, PK_R, PK_T) \mathrel{\widehat{=}}$$
$$receive.Msg1.C.R.tktag.pktag.tktag2.pktag2 \rightarrow$$
$$send.Msg2.R.T.Encrypt.SK_R.tag.tktag.pktag.tag2.tktag2.pktag2 \rightarrow$$
$$running2.R.C.tktag.pktag.tktag2.pktag2 \rightarrow$$
$$receive.Msg3.T.R.c1.c2 \rightarrow$$
$$send.Msg4.R.T.Encrypt.SK_R.c1.c2.r1.r2 \rightarrow SKIP$$

and the TGC is defined as:

$$TGC(SK_T, PK_T, PK_R, c1, c2) \mathrel{\widehat{=}}$$
$$receive.Msg2.R.T.Encrypt.SK_R.tag.tktag.pktag.tag2.tktag2.pktag2 \rightarrow$$
$$running.T.C.tktag.pktag.tktag2.pktag2 \rightarrow$$
$$send.Msg3.T.R.c1.c2 \rightarrow$$
$$receive.Msg4.R.T.Encrypt.SK_R.c1.c2.r1.r2 \rightarrow$$
$$send.Msg5.T.C.Encrypt.SK_T.PKItem_{item}.tktag.pktag.$$
$$\quad Encrypt.SK_T.PKItem_{item}.tktag2.pktag2 \rightarrow SKIP$$

A renaming is applied to the definitions of the customer, reseller, and the TGC to allow the adversary to intercept and fake messages on the communication channels. The input can come either from the channel $receive$ or from the channel $fake$ and the output can go to either $send$ or $intercept$. The customer, reseller, and TGC cannot tell whether a message has been sent from the other party or from the adversary.

$$CUSTOMER_0 \mathrel{\widehat{=}} CUSTOMER(item, pktag, pktag2, PK_T)$$
$$[[send.Msg1 \leftarrow send.Msg1, send.Msg1 \leftarrow intercept.Msg1,$$
$$receive.Msg5 \leftarrow receive.Msg5, receive.Msg5 \leftarrow fake.Msg5]]$$

$$RESELLER_0 \mathrel{\widehat{=}} RESELLER(item, tag, tag2, SK_R, PK_R, PK_T)$$
$$[[receive.Msg1 \leftarrow receive.Msg1, receive.Msg1 \leftarrow fake.Msg1,$$
$$send.Msg2 \leftarrow send.Msg2, send.Msg2 \leftarrow intercept.Msg2,$$
$$receive.Msg3 \leftarrow receive.Msg3, receive.Msg3 \leftarrow fake.Msg3,$$
$$send.Msg4 \leftarrow send.Msg4, send.Msg4 \leftarrow intercept.Msg4]]$$

$$TGC_0 \mathrel{\widehat{=}} TGC(SK_T, PK_T, PK_R)$$
$$[[receive.Msg2 \leftarrow receive.Msg2, receive.Msg2 \leftarrow fake.Msg2,$$
$$send.Msg3 \leftarrow send.Msg3, send.Msg3 \leftarrow intercept.Msg3,$$
$$receive.Msg4 \leftarrow receive.Msg4, receive.Msg4 \leftarrow fake.Msg4,$$
$$send.Msg5 \leftarrow send.Msg5, send.Msg5 \leftarrow intercept.Msg5]]$$

The Adversary

The adversary is parametrised by the information it knows. In our process the adversary is parametrised by $m1s$ which is the set of all message 1s the adversary knows, $m2s$ which is the set of all message 2s the adversary knows, $m3s$ which is the set of all message 3s the adversary knows, $m4s$ which is the set of all message 4s the adversary knows, $m5s$ which is the set of all message 5s the adversary knows, tk which is the set of all $Token$ the adversary knows, tg which is the set of all $Tags$ the adversary knows, at which is the set of all $ActualTags$ the adversary knows, c which is the set of all $Challenge$ the adversary knows, and r which is the set of all $Response$ the adversary knows. If the adversary knows the key to decrypt a message it learns the contents of the message, otherwise the adversary learns the encrypted content of the message.

The adversary can fake new messages using any values it has learnt, it can also fake messages by replaying the encrypted content of a message it has seen. The value $KnownKeys = \{SKItem_{item}, SK_C, SK_R, SK_T, SK_A, PK_A\}$ is defined as the keys the adversary knows. The CSP definition is:

$$
\begin{aligned}
&I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c, r) \mathrel{\widehat{=}} \\
&\quad receive.Msg1?a.b.tk'.tg'.tk2'.tg2' \rightarrow \\
&\qquad I(m1s, m2s, m3s, m4s, m5s, tk \cup \{tk', tk2'\}, tg \cup \{tg', tg2'\}, at, c, r) \\
&\quad \Box\ intercept.Msg1?a.b.tk'.tg'.tk2'.tg2' \rightarrow \\
&\qquad I(m1s, m2s, m3s, m4s, m5s, tk \cup \{tk', tk2'\}, tg \cup \{tg', tg2'\}, at, c, r) \\
&\quad \Box\ receive.Msg2?b.a.Encrypt.k.at'.tk'.tg'.at2'.tk2'.tg2' \rightarrow \\
&\qquad \text{if } k \in KnownKeys \text{ then } I(m1s, m2s, m3s, m4s, m5s, \\
&\qquad\qquad tk \cup \{tk', tk2'\}, tg \cup \{tg', tg2'\}, at \cup \{at', at2'\}, c, r) \\
&\qquad \text{else } I(m1s, m2s \cup \{Encrypt.k.at'.tk'.at2'.tk2'\}, m3s, m4s, m5s, \\
&\qquad\qquad tk, tg, at, c, r) \\
&\quad \Box\ intercept.Msg2?b.a.Encrypt.k.at'.tk'.tg'.at2'.tk2'.tg2' \rightarrow \\
&\qquad \text{if } k \in KnownKeys \text{ then } I(m1s, m2s, m3s, m4s, m5s, \\
&\qquad\qquad tk \cup \{tk', tk2'\}, tg \cup \{tg', tg2'\}, at \cup \{at', at2'\}, c, r) \\
&\qquad \text{else } I(m1s, m2s \cup \{Encrypt.k.at'.tk'.at2'.tk2'\}, m3s, m4s, m5s, \\
&\qquad\qquad tk, tg, at, c, r) \\
&\quad \Box\ receive.Msg3?a.b.c1'.c2' \rightarrow \\
&\qquad I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c \cup \{c1', c2'\}, r) \\
&\quad \Box\ intercept.Msg3?a.b.c1'.c2' \rightarrow \\
&\qquad I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c \cup \{c1', c2'\}, r) \\
&\quad \Box\ receive.Msg4?b.a.Encrypt.k.c1'c2'.r1'.r2' \rightarrow \\
&\qquad \text{if } k \in KnownKeys \text{ then } I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, \\
&\qquad\qquad c \cup \{c1', c2'\}, r \cup \{r1', r2'\}) \\
&\qquad \text{else } I(m1s, m2s, m3s, m4s \cup \{Encrypt.k.c1'c2'.r1'.r2'\}, m5s, \\
&\qquad\qquad tk, tg, at, c, r) \\
&\quad \Box\ intercept.Msg4?b.a.Encrypt.k.c1'c2'.r1'.r2' \rightarrow \\
&\qquad \text{if } k \in KnownKeys \text{ then } I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, \\
&\qquad\qquad c \cup \{c1', c2'\}, r \cup \{r1', r2'\}) \\
&\qquad \text{else } I(m1s, m2s, m3s, m4s \cup \{Encrypt.k.c1'c2'.r1'.r2'\}, m5s, \\
&\qquad\qquad tk, tg, at, c, r) \\
&\quad \Box\ receive.Msg5?b.a.Encrypt.k.pk.tk'.tg'.Encrypt.k.pk.tk2'.tg2' \rightarrow \\
&\qquad \text{if } k \in KnownKeys \text{ then } I(m1s, m2s, m3s, m4s, m5s,
\end{aligned}
$$

$$tk \cup \{tk', tk2'\}, tg \cup \{tg', tg2'\}, at, c, r)$$
else $I(m1s, m2s, m3s, m4s, m5s \cup \{Encrypt.k.pk.tk'.tg'.$
$$Encrypt.k.pk.tk2'.tg2'\}, tk, tg, at, c, r)$$
$\Box\ intercept.Msg5?b.a.Encrypt.k.pk.tk'.tg'.Encrypt.k.pk.tk2'.tg2' \rightarrow$
if $k \in KnownKeys$ then $I(m1s, m2s, m3s, m4s, m5s,$
$$tk \cup \{tk', tk2'\}, tg \cup \{tg', tg2'\}, at, c, r)$$
else $I(m1s, m2s, m3s, m4s, m5s \cup \{Encrypt.k.pk.tk'.tg'.$
$$Encrypt.k.pk.tk2'.tg2'\}, tk, tg, at, c, r)$$
$fake.Msg1?a.b?m : m1s \rightarrow I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c, r)$
$fake.Msg2?b.a?m : m2s \rightarrow I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c, r)$
$fake.Msg3?b.a?m : m3s \rightarrow I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c, r)$
$fake.Msg4?b.a?m : m4s \rightarrow I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c, r)$
$fake.Msg5?b.a?m : m5s \rightarrow I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c, r)$
$fake.Msg1?a.b?tk' : tk?tg' : tg?tk2' : tk?tg2' : tg \rightarrow$
$$I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c, r)$$
$fake.Msg2?a, b!Encrypt?k?at' : at?tk' : tk?at2' : at?tk2' : tk \rightarrow$
$$I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c, r)$$
$fake.Msg3?a.b?c1' : c?c2' : c \rightarrow I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c, r)$
$fake.Msg4?a.b!Encrypt?k?c1' : c?c2' : c?r1' : r?r2' : r \rightarrow$
$$I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c, r)$$
$fake.Msg5?a.b!Encrypt?k?pk?tk' : tk!Encrypt?k?pk?tk2' : tk \rightarrow$
$$I(m1s, m2s, m3s, m4s, m5s, tk, tg, at, c, r)$$

This model considers an adversary who has the initial knowledge of $pktagwrong$, $pktagwrong2$, $tagwrong$, $tagwrong2$, $tktag$, $tktag2$, $c1$, $c2$, $r1$, and $r2$:

$$INTRUDER \mathrel{\widehat{=}} I(\{\}, \{\}, \{\}, \{\}, \{\}, \{tktag, tktag2\}, \{pktagwrong, pktagwrong2\},$$
$$\{tagwrong, tagwrong2\}, \{c1, c2\}, \{r1, r2\})$$

Verification

The model of the protocol can now be put together with the intruder:

$$CHANS \mathrel{\widehat{=}} \{|send, receive|\}$$

$$AGENTS \mathrel{\widehat{=}} CUSTOMER_0 |[CHANS]| RESELLER_0 |[CHANS]| TGC_0$$

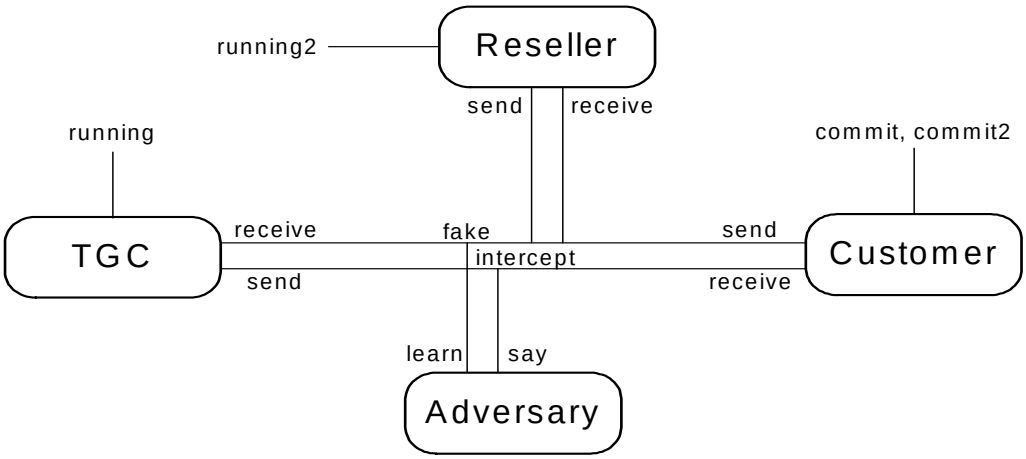

Figure A.5: CSP Network Model: Reseller Generating a Tag

$$SYSTEM \mathrel{\widehat{=}} AGENTS|[\{|send, receive, fake, intercept|\}]|INTRUDER$$

Figure A.5 illustrates the processes and channels in the model of the protocol. The customer, reseller, and the TGC processes have access to the channels $send$ and $receive$. The TGC has access to the channel $running$, the reseller has access to the channel $running2$ and the customer has access to the channels $commit$ and $commit2$. The adversary has access to the channels $fake$ and $intercept$.

FDR takes as input an implementation of the protocol and a specification and checks that the implementation refines the specification. This phase of the protocol has two specifications for the reseller generating the tag. The first specification prevents replay. Informally, if the customer receives a tag then two different values must have been signed by the TGC. In CSP we the specification is defined as:

$$SPEC_0 \mathrel{\widehat{=}}$$
$$running.T.C.tktag.pktag.tktag2.pktag2 \rightarrow$$
$$commit.C?T.tktag.pktag.tktag2.pktag2 \rightarrow SPEC$$

$$SPEC_1 \mathrel{\widehat{=}}$$
$$\{|running.T.C.tktag.pktag.tktag2.pktag2,$$
$$commit.C.T.tktag.pktag.tktag2.pktag2|\}$$

$$SPEC \mathrel{\widehat{=}} SPEC_0 \,|||\, RUN(\Sigma \setminus SPEC_1)$$

The term Σ is the set of all possible events. The FDR model checker checks $SPEC \sqsubseteq_{\mathrm{T}} SYSTEM$ and returns TRUE after 457 states with 2005 transitions.

The second specification checks for a network sniffing attack. Informally, if a customer receives tags signed by the TGC then the reseller must have sent a message to the TGC to request the generation of the tags. In CSP the specification is defined as:

$$SPEC2_0 \mathrel{\widehat{=}}$$
$$running2.R.C.tktag.pktag.tktag2.pktag2 \rightarrow$$
$$commit2.C?R.tktag.pktag.tktag2.pktag2 \rightarrow SPEC$$

$$SPEC2_1 \mathrel{\widehat{=}}$$
$$\{|running2.R.C.tktag.pktag.tktag2.pktag2,$$
$$commit2.C.R.tktag.pktag.tktag2.pktag2|\}$$

$$SPEC2 \mathrel{\widehat{=}} SPEC2_0 \,|||\, RUN(\Sigma \setminus SPEC2_1)$$

The term Σ is the set of all possible events. The FDR model checker checks $SPEC2 \sqsubseteq_{\mathrm{T}} SYSTEM$ and returns TRUE after 1423 states with 4835 transitions.

References

[1] 4FRIENDSONLY. New technologies for virtual goods, 2010. http://www.4fo.de/en/potato.htm.

[2] ABELSON, H., ADIDA, B., LINKSVAYER, M., AND YERGLER, N. ccREL: The creative commons rights expression langauge. Tech. rep., Creative Commons, March 2008.

[3] AIKEN, A., CHEN, J., STONEBRAKER, M., AND WOODRUFF, A. Tioga-2: a direct manipulation database visualization environment. In *Proceedings of the Twelfth International Conference on Data Engineering, 1996.* (Feb-1 Mar 1996), pp. 208 –217.

[4] APPLE. iTunes store top music retailer in the us. http://www.apple.com/pr/library/2008/04/03itunes.html.

[5] APPLE. iTunes store tops 10 billion songs sold. http://www.apple.com/pr/library/2010/02/25itunes.html.

[6] BELLARE, M., AND ROGAWAY, P. Optimal asymmetric encryption. In *Proceedings of the 13th International Conference on the Theory and Applications of Cryptographic Techniques EUROCRYPT'94* (1994), pp. 92–111.

[7] BELLARE, M., AND ROGAWAY, P. The exact security of digital signatures-how to sign with RSA and Rabin. In *Proceedings of the 15th annual international conference on Theory and application of cryptographic techniques* (Berlin, Heidelberg, 1996), EUROCRYPT'96, Springer-Verlag, pp. 399–416.

[8] BELLARE, M., AND ROGAWAY, P. The exact security of digital signatures-how to sign with RSA and Rabin. In *Proceedings of the 15th annual international conference on Theory and application of cryptographic techniques EUROCRYPT'96* (Berlin, Heidelberg, 1996), Springer-Verlag, pp. 399–416.

[9] BENDER, W., GRUHL, D., MORIMOTO, N., AND LU, A. Techniques for data hiding. *IBM Systems Journal 35*, 3.4 (1996), 313 –336.

[10] BOLAND, F., O'RUANAIDH, J., AND DAUTZENBERG, C. Watermarking digital images for copyright protection. In *Proceedings of the 5th International Conference on Image Processing and its Applications* (Jul 1995), pp. 326 –330.

[11] BRANDS, S. Untraceable off-line cash in wallets with observers (extended abstract). In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 1994), CRYPTO '93, Springer-Verlag, pp. 302–318.

[12] BRAUN, U., SHINNAR, A., AND SELTZER, M. Securing provenance. In *The 3rd USENIX Workshop on Hot Topics in Security* (Berkeley, CA, USA, July 2008), USENIX HotSec, USENIX Association, pp. 1–5.

[13] CAMENISCH, J., AND HERREWEGHEN, E. V. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM conference on Computer and communications security CCS '02* (New York, NY, USA, 2002), ACM, pp. 21–30.

[14] CAMENISCH, J., HOHENBERGER, S., AND LYSYANSKAYA, A. Compact e-cash. In *Proceedings of the 24th annual international conference on Theory and application of cryptographic techniques EUROCRYPT'05* (2005), pp. 302–321.

[15] CAMENISCH, J., AND LYSYANSKAYA, A. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques EUROCRYPT '01* (London, UK, 2001), Springer-Verlag, pp. 93–118.

[16] CAMENISCH, J., AND LYSYANSKAYA, A. A signature scheme with efficient protocols. In *Proceedings of the 3rd international conference on Security in communication networks* (Berlin, Heidelberg, 2003), SCN'02, Springer-Verlag, pp. 268–289.

[17] CAMENISCH, J., AND MICHELS, M. Proving in zero-knowledge that a number is the product of two safe primes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques EUROCRYPT'99* (Berlin, Heidelberg, 1999), Springer-Verlag, pp. 107–122.

[18] CHAUM, D. Blind signatures for untraceable payments. In *Proceedings on Advances in cryptology CRYPTO '82* (1982), pp. 199–203.

[19] CHAUM, D. Blind signature system. In *Proceedings on Advances in cryptology CRYPTO '83* (1983), p. 153.

[20] CHAUM, D., FIAT, A., AND NAOR, M. Untraceable electronic cash. In *Proceedings on Advances in cryptology CRYPTO '88* (New York, NY, USA, 1990), Springer-Verlag New York, Inc., pp. 319–327.

[21] CHAUM, D. L. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM 24*, 2 (1981), 84–90.

[22] CHEBOTKO, A., CHANG, S., LU, S., FOTOUHI, F., AND YANG, P. Scientific workflow provenance querying with security views. In *Proceedings of the International Conference on Web-Age Information Management* (Los Alamitos, CA, USA, 2008), IEEE Computer Society, pp. 349–356.

[23] CORON, J.-S., JOYE, M., NACCACHE, D., AND PAILLIER, P. Universal padding schemes for RSA. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology* (London, UK, UK, 2002), CRYPTO '02, Springer-Verlag, pp. 226–241.

[24] COX, I. J., KILIAN, J., LEIGHTON, F. T., AND SHAMOON, T. A secure, robust watermark for multimedia. In *Proceedings of the First International Workshop on Information Hiding* (London, UK, 1996), Springer-Verlag, pp. 185–206.

[25] CRAMER, R., AND SHOUP, V. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 1998), Springer-Verlag, pp. 13–25.

[26] D., G., A., L., AND W., B. Fine-grained tracking of grid infections. In *Proceedings of the First International Workshop on Information Hiding* (1996), pp. 295–316.

[27] DESMEDT, Y., AND FRANKEL, Y. Shared generation of authenticators and signatures (extended abstract). In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 1992), CRYPTO '91, Springer-Verlag, pp. 457–469.

[28] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *In Proceedings of the 13th USENIX Security Symposium* (2004), pp. 303–320.

[29] DINH, T. T. A., AND RYAN, M. Verifying security property of peer-to-peer systems using csp. In *Proceedings of the 15th European conference on Research in computer security* (Berlin, Heidelberg, 2010), ESORICS'10, Springer-Verlag, pp. 319–339.

[30] DOLEV, D., AND YAO, A. C. On the security of public key protocols. *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science 0* (1981), 350–357.

[31] DURFEE, G., AND FRANKLIN, M. Distribution chain security. In *Proceedings of the 7th ACM conference on Computer and communications security CCS '00* (New York, NY, USA, 2000), ACM, pp. 63–70.

[32] EASTLAKE, D., AND JONES, P. RFC 3174: US secure hash algorithm 1 (SHA1), September 2001.

[33] FORMAL SYSTEMS (EUROPE) LTD. Fdr2 user manual. `http://www.fsel.com/documentation/fdr2/html/index.html`.

[34] FOSTER, I., VOCKLER, J., WILDE, M., AND ZHAO, Y. Chimera: a virtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management* (2002), pp. 37 – 46.

[35] FUJISAKI, E., AND OKAMOTO, T. How to enhance the security of public-key encryption at minimum cost. In *Proceedings of the Second International Workshop on Practice and Theory in Public Key Cryptography* (London, UK, 1999), PKC '99, Springer-Verlag, pp. 53–68.

[36] FUJISAKI, E., OKAMOTO, T., POINTCHEVAL, D., AND STERN, J. RSA-OAEP is secure under the rsa assumption. *Journal of Cryptology 17* (March 2004), 81–104.

[37] GAMAL, T. E. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings on Advances in cryptology CRYPTO 84* (New York, NY, USA, 1985), Springer-Verlag New York, Inc., pp. 10–18.

[38] GEHANI, A., BAIG, B., MAHMOOD, S., TARIQ, D., AND ZAFFAR, F. Fine-grained tracking of grid infections. In *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing (GRID), 2010* (oct. 2010), pp. 73 –80.

[39] GROTH, P., MILES, S., AND MOREAU, L. A model of process documentation to determine provenance in mash-ups. *ACM Trans. Internet Technol. 9* (February 2009), 3:1–3:31.

[40] GROTH, P., AND MOREAU, L. Representing distributed systems using the open provenance model. *Future Gener. Comput. Syst. 27* (June 2011), 757–765.

[41] HAMM, S., AND TUCKER, M. How secure is your domain? *Business Week* (March 2007).

[42] HARTUNG, F., AND GIROD, B. Watermarking of uncompressed and compressed video. *Signal Process. 66* (May 1998), 283–301.

[43] HASAN, R., SION, R., AND WINSLETT, M. The case of the fake Picasso: preventing history forgery with secure provenance. In *Proccedings of the 7th conference on File and storage technologies* (Berkeley, CA, USA, 2009), USENIX Association, pp. 1–14.

[44] HASAN, R., SION, R., AND WINSLETT, M. Preventing history forgery with secure provenance. *Trans. Storage 5* (December 2009), 12:1–12:43.

[45] HENRY, R., HENRY, K., AND GOLDBERG, I. Making a nymbler nymble using verbs. In *Proceedings of the 10th international conference on Privacy enhancing technologies* (Berlin, Heidelberg, 2010), PETS'10, Springer-Verlag, pp. 111–129.

[46] HOARE, C. A. R. Communicating sequential processes. *Commun. ACM 21*, 8 (1978), 666–677.

[47] HSU, C.-L., WU, T.-S., AND WU, T.-C. Improvements of generalization of threshold signature and authenticated encryption for group communications. *Inf. Process. Lett. 81*, 1 (2002), 41–45.

[48] HUI, M. L., AND LOWE, G. Safe simplifying transformations for security protocols or not just the Needham Schroeder public key protocol. In *Proceed-*

ings of the 12th IEEE workshop on Computer Security Foundations (Washington, DC, USA, 1999), CSFW '99, IEEE Computer Society, pp. 32–77.

[49] ICANN. Attention registrants: Registerfly pending termination. `http://www.icann.org/en/announcements/announcement-registerfly.htm`.

[50] ISO. Information technology - security techniques. `http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html`.

[51] ISO 21000-3:2003. *Information Technology - Multimedia Framework (MPEG-21) - Part 3: Digital Item Identification*. ISO, Geneva, Switzerland.

[52] ISO 21000-5:2004. *Information Technology - Multimedia Framework (MPEG-21) - Part 5: Rights Expression Language*. ISO, Geneva, Switzerland.

[53] LANTER, D. P. Design of a lineage-based meta-data base for gis. *Cartography And Geographic Information Systems 18*, 4 (1991), 255–261.

[54] LEE, W.-B., AND CHANG, C.-C. (t, n) threshold digital signature with traceability property. *J. Inf. Sci. Eng.* (1999), 669–678.

[55] LOWE, G. Breaking and fixing the Needham-Schroeder public-key protocol using fdr. In *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems* (London, UK, 1996), Springer-Verlag, pp. 147–166.

[56] LOWE, G. Casper: a compiler for the analysis of security protocols. *J. Comput. Secur. 6*, 1-2 (1998), 53–84.

[57] LOWE, G., AND ROSCOE, B. Using CSP to detect errors in the TMN protocol. *IEEE Trans. Softw. Eng. 23* (October 1997), 659–669.

[58] MANUEL, C., TSUTOMU, M., AND HIDEKI, I. Efficient and secure multi-party generation of digital signatures based on discrete logarithms (special section on discrete mathematics and its applications). *IEICE transactions on fundamentals of electronics, communications and computer sciences 76*, 4 (1993-04-25), 532–545.

[59] MOREAU, L., CLIFFORD, B., FREIRE, J., FUTRELLE, J., GIL, Y., GROTH, P., KWASNIKOWSKA, N., MILES, S., MISSIER, P., MYERS, J., PLALE, B.,

SIMMHAN, Y., STEPHAN, E., AND DEN BUSSCHE, J. V. The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems* (July 2010).

[60] MORI, R., AND KAWAHARA, M. Superdistribution: An electronic infrastructure for the economy of the future. *Transactions of the Information Processing Society of Japan 38*, 7 (1997), 1465–1472.

[61] NAIR, S. K., GERRITS, R., CRISPO, B., AND TANENBAUM, A. S. Turning teenagers into stores. *Computer 41*, 2 (2008), 58–62.

[62] NAIR, S. K., POPESCU, B. C., GAMAGE, C., CRISPO, B., AND TANENBAUM, A. S. Enabling DRM-preserving digital content redistribution. In *Proceedings of the Seventh IEEE International Conference on E-Commerce Technology CEC '05* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 151–158.

[63] NIST COMPUTER SECURITY DIVISION. Federal information processing standards 186-3. Tech. rep., NIST Computer Security Division, July 2009.

[64] NUTZEL, J., AND GRIMM, R. Potato system and signed media format - an alternative approach to online music business. *Proceedings of the 3rd International Conference on Web Delivering of Music WEDELMUSIC'03* (Sept. 2003), 23–26.

[65] ODRL INITIATIVE. ODRL v2.0 - core specification, February 2011. `http://odrl.net/2.0/DS-ODRL-Model.html`.

[66] OPEN MOBILE ALLIANCE. DRM architecture OMA. Tech. rep., Open Mobile Allicance, 2008.

[67] P1817, I. W. G. Initial technical description of the p1817 standard. Tech. rep., IEEE, 2010.

[68] PALMER, B., BUBENDORFER, K., AND WELCH, I. A protocol for anonymously establishing digital provenance in reseller chains. In *to appear Proceedings of the 15th International Financial Cryptography Conference (FC-2011)* (Jan 2011).

[69] PALMER, B., BUBENDORFER, K., AND WELCH, I. Verifying digital provenance in web services. In *to appear Proceedings of the 4th International*

IEEE/ACM Conference on Utility and Cloud Computing. Cloud Service Security and Quality Management Workshop (Dec 2011).

[70] PARK, C., AND KUROSAWA, K. New El-Gamal type threshold digital signature scheme. *IEICE transactions on fundamentals of electronics, communications and computer sciences 79*, 1 (January 1996), 86–93.

[71] PATSAKIS, C., AND ALEXANDRIS, N. Multimedia information security. In *Multimedia Services in Intelligent Environments*, G. Tsihrintzis and L. Jain, Eds., vol. 120 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg, 2008, pp. 257–273.

[72] PFITZMANN, A., AND KÖHNTOPP, M. Anonymity, unobservability, and pseudeonymity — a proposal for terminology. In *International workshop on Designing privacy enhancing technologies* (New York, NY, USA, 2001), Springer-Verlag New York, Inc., pp. 1–9.

[73] POINTCHEVAL, D. Chosen-ciphertext security for any one-way cryptosystem. In *Proceedings of the 3rd International Workshop on Practice and Theory in Public Key Cryptography, PKC 2000, Melbourne, Victoria, Australia, January 18-20, 2000* (2000), pp. 129–146.

[74] POINTCHEVAL, D., AND STERN, J. Security proofs for signature schemes. In *Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology EUROCRYPT '96* (1996), Springer-Verlag, pp. 387–398.

[75] POTATO SYSTEM. Potato system - about us, 2010. `http://www.potatosystem.com/info/en/imprint`.

[76] RIVEST, R. RFC 1321: The MD5 message-digest algorithm, April 1992.

[77] RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21* (February 1978), 120–126.

[78] ROSCOE, A. W. Model-checking CSP. In *A classical mind: essays in honour of C. A. R. Hoare*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1994, pp. 353–378.

[79] ROSCOE, A. W. CSP and determinism in security modelling. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 1995), SP '95, IEEE Computer Society, pp. 114–.

[80] RSA LABORATORIES. Public-key cryptography standards (PKCS): RSA cryptography specification version 2.1. Tech. rep., RSA Laboratories, June 2002.

[81] RYAN, P., AND SCHNEIDER, S. *The modelling and analysis of security protocols: the csp approach*, first ed. Addison-Wesley Professional, 2000.

[82] SCHMIDT, A. U. On the superdistribution of digital goods. *Invited paper at the 2008 International Workshop on Multimedia Security in Communication (MUSIC'08)* (2008).

[83] SCHNORR, C. P. Efficient identification and signatures for smart cards. In *Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology EUROCRYPT '89* (New York, NY, USA, 1990), Springer-Verlag New York, Inc., pp. 688–689.

[84] SEITZ, J. *Digital Watermarking For Digital Media*. Information Resources Press, Arlington, VA, USA, 2005.

[85] SERBAN, C., CHEN, Y., ZHANG, W., AND MINSKY, N. The concept of decentralized and secure electronic marketplace. *Electronic Commerce Research 8*, 1-2 (2008), 79–101.

[86] SHOUP, V. Oaep reconsidered. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 2001), CRYPTO '01, Springer-Verlag, pp. 239–259.

[87] STEVENS, R., ROBINSON, A., AND GOBLE, C. mygrid: personalised bioinformatics on the information grid. *Bioinformatics 19 Suppl 1* (2003), i302–4.

[88] SYVERSON, P. F., GOLDSCHLAG, D. M., AND REED, M. G. Anonymous connections and onion routing. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy SP '97* (Washington, DC, USA, 1997), IEEE Computer Society, pp. 44–56.

[89] TAN, V., GROTH, P. T., MILES, S., JIANG, S., MUNROE, S., TSASAKOU, S., AND MOREAU, L. Security issues in a SOA-based provenance system. In *IPAW* (2006), pp. 203–211.

[90] VAN SCHYNDEL, R. G., TIRKEL, A. Z., AND OSBORNE, C. F. A digital watermark. In *Proceedings of the International Conference on Image Processing 1994* (1994), vol. 2, pp. 86–90.

[91] WOODRUFF, A., AND STONEBRAKER, M. Supporting fine-grained data lineage in a database visualization environment. In *Proceedings of the Thirteenth International Conference on Data Engineering* (Washington, DC, USA, 1997), ICDE '97, IEEE Computer Society, pp. 91–102.

[92] ZHAO, J., GOBLE, C., STEVENS, R., AND BECHHOFER, S. Semantically linking and browsing provenance logs for e-science. In *Semantics of a Networked World*, M. Bouzeghoub, C. Goble, V. Kashyap, and S. Spaccapietra, Eds., vol. 3226 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2004, pp. 158–176.