A Secure Infrastructural Strategy for
Safe Autonomous Mobile Agents

by

Michelangelo Antonio Giansiracusa

Bachelor of Information Technology
(Honours)
First Class Honours

Thesis submitted in accordance with the regulations for
Degree of Doctor of Philosophy

Information Security Institute
Faculty of Information Technology
Queensland University of Technology

27 July 2005
Keywords

MASHIn, MASHs, RECDAM, SCC4MAP, application-independent mobile agents, autonomous mobile agents, checking hosts, confidence scores, distributed systems, dummy callback classes, first-hand observations, itinerant agents, language-independent mobile agents, mobile agent execution integrity, mobile agent paradigm, mobile agent platform, mobile agent platform owner, mobile agent privacy, mobile agent security, mobile agent user, mobile agents, mobile computing, real callback classes, real-time in-mission checking, reference state, reputation, second-hand opinions, secure callback classes, trusted third party, ubiquitous framework.
Abstract

Portable languages and distributed paradigms have driven a wave of new applications and processing models. One of the most promising, certainly from its early marketing, but disappointing (from its limited uptake) is the mobile agent execution and data processing model. Mobile agents are autonomous programs which can move around a heterogeneous network such as the Internet, crossing through a number of different security domains, and perform some work at each visited destination as partial completion of a mission for their agent user.

Despite their promise as a technology and paradigm to drive global electronic services (i.e. any Internet-driven-and-delivered service, not solely e-commerce related activities), their uptake on the Internet has been very limited. Chief among the reasons for the paradigm’s practical under-achievement is there is no ubiquitous framework for using Internet mobile agents, and non-trivial security concerns abound for the two major stakeholders (mobile agent users and mobile agent platform owners).

While both stakeholders have security concerns with the dangers of the mobile agent processing model, most investigators in the field are of the opinion that protecting mobile agents from malicious agent platforms is more problematic than protecting agent platforms from malicious mobile agents. Traditional cryptographic mechanisms are not well-suited to counter the bulk of the threats associated with the mobile agent paradigm due to the untrusted hosting of an agent and its intended autonomous, flexible movement and processing.

In our investigation, we identified that the large majority of the research undertaken on mobile agent security to date has taken a micro-level perspective. By this we mean research focused solely on either of the two major stakeholders, and even then often only on improving measures to address one security issue dear to the stakeholder - for example mobile agent privacy (for agent users) or access control to platform resources (for mobile agent platform owners).
We decided to take a more encompassing, higher-level approach in tackling mobile agent security issues. In this endeavour, we developed the beginnings of an infrastructural-approach to not only reduce the security concerns of both major stakeholders, but bring them transparently to a working relationship. Strategic utilisation of both existing distributed system trusted-third parties (TTPs) and novel mobile agent paradigm-specific TTPs are fundamental in the infrastructural framework we have devised.

Besides designing an application and language independent framework for supporting a large-scale Internet mobile agent network, our Mobile Agent Secure Hub Infrastructure (MASHIn) proposal encompasses support for flexible access control to agent platform resources. A reliable means to track the location and processing times of autonomous Internet mobile agents is discussed, with fault-tolerant handling support to workaround unexpected processing delays. Secure, highly-effective (in comparison to existing mechanisms) strategies for providing mobile agent privacy, execution integrity, and stakeholder confidence scores were devised - all which fit comfortably within the MASHIn framework. We have deliberately considered the interests - without bias - of both stakeholders when designing our solutions.

In relation to mobile agent execution integrity, we devised a new criteria for assessing the robustness of existing execution integrity schemes. Whilst none of the existing schemes analysed met a large number of our desired properties for a robust scheme, we identified that the objectives of Hohl’s reference states scheme were most admirable - particularly real-time in-mission execution integrity checking. Subsequently, we revised Hohl’s reference states protocols to fit in the MASHIn framework, and were able to overcome not only the two major limitations identified in his scheme, but also meet all of our desired properties for a robust execution integrity scheme (given an acceptable decrease in processing efficiency).

The MASHIn offers a promising new perspective for future mobile agent security research and indeed a new framework for enabling safe and autonomous Internet mobile agents. Just as an economy cannot thrive without diligent care given to micro and macro-level issues, we do not see the security prospects of mobile agents (and ultimately the prospects of the mobile agent paradigm) advancing without diligent research on both levels.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keywords</td>
<td>iii</td>
</tr>
<tr>
<td>Abstract</td>
<td>v</td>
</tr>
<tr>
<td>Declaration</td>
<td>xix</td>
</tr>
<tr>
<td>Previously Published Material</td>
<td>xxi</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>xxiii</td>
</tr>
<tr>
<td>1 Thesis Overview</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Contributions Encapsulated in the Thesis</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Thesis Chapter Outline</td>
<td>3</td>
</tr>
<tr>
<td>2 Introducing the Mobile Agent Paradigm</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Mobile Agents Defined</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Mobile Agent Constituent Parts</td>
<td>9</td>
</tr>
<tr>
<td>2.3 The Life-cycle of a Mobile Agent</td>
<td>10</td>
</tr>
<tr>
<td>2.4 The Evolution of Mobile Agents</td>
<td>12</td>
</tr>
<tr>
<td>2.4.1 Client-Server (CS) Paradigm</td>
<td>13</td>
</tr>
<tr>
<td>2.4.2 Remote Evaluation (REV) Paradigm</td>
<td>13</td>
</tr>
<tr>
<td>2.4.3 Code On Demand (COD) Paradigm</td>
<td>14</td>
</tr>
<tr>
<td>2.4.4 Mobile Agent (MA) Paradigm</td>
<td>15</td>
</tr>
<tr>
<td>2.4.5 Historically Significant Mobile Agent Systems</td>
<td>16</td>
</tr>
<tr>
<td>2.4.6 Mobile Agents Today</td>
<td>17</td>
</tr>
<tr>
<td>2.5 Applications Geared Towards Mobile Agent Technology</td>
<td>18</td>
</tr>
<tr>
<td>2.6 Factors Limiting the Widespread Adoption of Mobile Agents</td>
<td>19</td>
</tr>
<tr>
<td>2.7 Summary</td>
<td>21</td>
</tr>
</tbody>
</table>
3 Security Problems with the Internet Mobile Agent Paradigm

3.1 Fundamental Security Principles of Distributed Systems

3.1.1 Authentication

3.1.2 Message Confidentiality

3.1.3 Message Integrity

3.1.4 Accountability and Message Non-repudiation

3.1.5 Anonymity

3.1.6 Availability and Denial-of-Service

3.2 Avenues of Threats in a Mobile Agent Network

3.3 Threats to Mobile Agent Platform Owners

3.3.1 Denial-of-Service from Agents

3.3.2 Mobile Agent Gains Unauthorised Access

3.3.3 Agent Refusal to Pay for Services Delivered to It

3.3.4 Infection via Malicious Agent Software Forms

3.3.5 Attacks Against Inter-agent or Platform Communications

3.4 Threats to Mobile Agent Users

3.4.1 Impersonation from an Interacting Agent or Agent Platform

3.4.2 Agent Platform Denies Proper Service to Agent

3.4.3 Agent Platform Obtains Private Data or Mines the Agent

3.4.4 Eavesdropping of Agent Messages

3.4.5 Manipulation of Inter-agent Messages

3.4.6 Agent Platform Modifies the Agent’s Code or State

3.4.7 Agent Platform Manipulates Agent’s Control Flow

3.4.8 Agent Platform Misroutes the Agent

3.4.9 Replay Execution Attack on an Agent

3.4.10 Agent Platform Repudiation of Communication

3.4.11 Agent Platform Repudiation of Execution

3.4.12 Agent Platform Repudiation of Agent Migration

3.4.13 Agent Platform Returns Wrong Results

3.5 Summary

4 Existing Strategies for Protecting Major Mobile Agent System Stakeholders

4.1 Countermeasures for Thwarting Threats to Mobile Agent Platforms

4.1.1 Software-based Fault Isolation and Sandboxing

4.1.2 Safe Code Interpretation and Using Type Safe Languages
6.3.3 Credential Infrastructure Component in the MASHIn Model

6.4 MASHIn Authentication and Anonimisation Processes

6.5 MASHIn Agent Location and Itinerary Event Monitoring Processes

6.6 Summary

7 Protecting Mobile Agent Data and Code Privacy

7.1 The Objectives of Mobile Agent Data And Code Privacy

7.2 Existing Strategies to Achieving Mobile Agent Privacy

7.2.1 Approaches Which Make the Adversary’s Goal More Difficult

7.2.1.1 Obfuscation Strategies

7.2.1.2 Environmental Key Generation Towards Clueless Agents

7.2.2 Approaches Which Solve the General Case Adversary Complexity

7.2.2.1 Computing with Encrypted Functions

7.2.2.2 Secure Computation Service

7.2.2.3 Code Splitting Among Cooperating Agents

7.2.2.4 Supervisor-Worker Framework

7.2.3 Summary of Existing Mobile Agent Privacy Strategies

7.3 Secure Callback Classes for Mobile Agent Privacy

7.3.1 Rationale for Strategy

7.3.2 Strategy Overview

7.3.3 Proof-of-Concept Prototype Application

7.3.4 Extension for Forwarding ‘Real’ Callback Classes

7.3.5 Protecting the Privacy of Agent Platform Inputs

7.4 SCC4MAP Comparison with Existing Strategies

7.5 Summary

8 Protecting the Execution Integrity of Mobile Agents

8.1 Execution Integrity Objectives

8.1.1 Quality Research Paper/Author Finder Mobile Agent Application

8.1.2 Properties of Interest

8.2 Review of Existing Strategies

8.2.1 Partial Result Encapsulation

8.2.2 Holographic Proofs
9.2.3 Security Measures ................................................. 211
9.2.4 Formula for Calculating Agent Platform Confidence Scores 212
9.3 Anonymous Application of Agent User Properties for Protecting Agent Platforms ................................................. 214
  9.3.1 Pivotal Role of MASHs ........................................... 214
  9.3.2 MASHIn Reputation Example Scenario ......................... 215
  9.3.3 Meeting Our Aims ............................................... 217
  9.3.4 Fee-Charge Structure .......................................... 219
9.4 Flexibility to Consider Second-Hand Opinions .................. 219
  9.4.1 Credit Rating Agencies in the Real-World .................... 220
  9.4.2 How the MASHIn’s Confidence Scores Fare .................. 221
  9.4.3 The Option to Add-in Second-Hand Opinions ................. 222
9.5 Summary .................................................................. 223

10 Measures to Consider in Protecting MASHs and Checking Hosts 227
  10.1 Distributed Deployment and Backup ............................. 228
  10.2 Mandatory Access Control and Secure Operating Systems .... 230
  10.3 Virtual Private Networking ..................................... 234
  10.4 Defence-in-depth Firewall Design ................................ 238
  10.5 DDoS Countermeasures .......................................... 240
  10.6 Trusted Computing Initiatives ................................... 243
  10.7 Summary ................................................................ 245

11 Implementation Recommendations .................................. 247
  11.1 Technical Considerations ........................................ 248
  11.2 Execution Checking ............................................. 251
  11.3 Confidence Scores ................................................ 253
  11.4 Scalability of the MASHIn ...................................... 254
  11.5 Application Case Scenarios ..................................... 256
    11.5.1 Purchase Shopping Mobile Agents ......................... 256
    11.5.2 Group Meeting Negotiator ................................. 257
    11.5.3 Fairer Research Paper Reviewer ......................... 259
  11.6 Summary .......................................................... 261

12 Conclusions and Avenues for Future Work .......................... 263
  12.1 Contributions ..................................................... 264
List of Figures
2.1
2.2
2.3
2.4
2.5

Mobile agent state diagram (adapted from [233]).
Client-server paradigm pictured. . . . . . . . . . .
Remote evaluation paradigm pictured. . . . . . .
Code on demand paradigm pictured. . . . . . . .
Mobile agent paradigm pictured. . . . . . . . . . .

3.1
3.2
3.3
3.4
3.5

Authentication problem/solution in a distributed system. . . . . .
Message confidentiality problem/solution in a distributed system.
Message integrity problem/solution in a distributed system. . . .
Message non-repudiation problem/solution in a distributed system.
Fundamental agent system model (adapted from [96]). . . . . . .

25
26
27
29
31

5.1

The SOMA mobile agent system privilege management model centralised by domain (adapted from [43]). . . . . . . . . . . . . . . .

78

6.1
6.2
6.3
6.4

6.5
6.6

7.1

.
.
.
.
.

.
.
.
.
.

.
.
.
.
.

.
.
.
.
.

.
.
.
.
.

.
.
.
.
.

.
.
.
.
.

.
.
.
.
.

.
.
.
.
.

Major MASHIn software modules. . . . . . . . . . . . . . . . . . .
MASHIn RA, CA and major stakeholder responsibilities and interactions (adapted from [113]). . . . . . . . . . . . . . . . . . . .
ITU’s X.509 Attribute Certificate content elements (derived from
[90]). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
MASHIn credential infrastructure policy components and inputs
for determining an agent’s localised access rights to an agent platform’s resources. . . . . . . . . . . . . . . . . . . . . . . . . . . .
Considering MASHIn agent (user) authentication and anonimisation processes. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
MASHIn agent location tracking and TTL processing example scenario. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

11
13
14
14
16

101
107
113

114
117
123

Purchase agent buys the best-priced Gizmo. . . . . . . . . . . . . 129

xv


# List of Tables

3.1 Summary of noted threats, general-case mitigation feasibility, and whether our proposal offers a solution to thwarting the threat. . . 42

4.1 Summary of countermeasures coverage to threats (1 of 2). . . . 71
4.2 Summary of countermeasures coverage to threats (2 of 2). . . . 72

7.1 Summary analysis of existing approaches supporting mobile agent privacy. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139

8.1 Appraisal of mechanisms for providing robust mobile agent execution integrity. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 170
8.2 A comparison of existing mechanisms for mobile agent execution integrity with our RECDAM scheme. . . . . . . . . . . . . . 196

11.1 Mobile agent research systems with strong security perspective. . 250
Declaration

The work contained in this thesis has not been previously submitted for a degree or diploma at any higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signed: ___________________________ Date: ___________________________
Previously Published Material

The following papers have been published or presented, and contain material based on the content of this thesis.


Acknowledgements

In this tiny space, I get to thank those special people who have helped make this life ambition possible. I start (and will always end) with the two most important reasons why I live, breathe, and strive to meet and beat (to the best of my ability) life’s many challenges - my enduring, strong, remarkable parents. Mum: if it was not for your friendship, understanding and strength-of-character, I certainly would not have had the courage to strive and achieve a fraction of what I have managed. You are, and always will be, my unrivalled number one friend. Dad: you have an unsurpassable mentality of applying yourself to your work, and an amazing attitude to never give in - no matter how tricky a task becomes - that has and continues to inspire me. I thank you both for your never-ending, brilliant love and for believing a formal education - coincidentally that you both largely missed out on - would be the best direction in life that I could take.

To my PhD supervisors... Selwyn Russell: as my principal supervisor, I thank you for your friendship and for always believing I was better than that I considered myself to be. The level of autonomy you granted me was much appreciated, allowing me to be both productive and creative. Andrew Clark: your professionalism, suggestions on how to improve my writing, and neat tips for typesetting LaTeX documents is appreciated. John Hynd: your attention to detail, advice on viewing things with increased rationality, and leadership to strive to do better (in the years before I was a PhD student, and since about a year ago when you enthusiastically agreed to join my PhD supervision team) is to be applauded.

To some current and former members of my research centre... Rupinder Gill and Andrew Marrington were welcome buddies in the last year of my candidature - thank you for your friendship, and the very best of wishes for your Doctoral thesis productions in the next couple of years. Minna Yao, who produced a Doctoral thesis in the mobile agent security domain, even though her
and my refined field of speciality and modus operandi were like chalk and cheese: thank you for your encouragement, and for never shying away from saying hello to me.

I want to blow my own trumpet a little now... I have always been an individual, an enigmatic character, and at times my own worst enemy. I feel grateful that I undertook the PhD journey because it has made me more comfortable with me and my limitations, allowing me to gain a better perspective on life and showing pride in my work. What is most gratifying in completing this thesis is the following undeniable fact: The whole thesis process - from topic selection and refinement, background research, conceptual ideas, contribution, and writing - was completed on my own, with no assistance from Honours students, co-researchers, or co-writers (besides some basic - but nevertheless expert - editorial suggestions from my supervisors). I set out with the ambition of achieving a PhD on my own back, my way. I prefer to live and die on my own sword, knowing what ever I achieve I can be genuinely proud of. Thus, I am very proud of this piece of work, and no one can dispute my achievement.

To the federal government, QUT, and the ISI (formerly ISRC)... Thank you for your scholarships and stipend top-ups which provided me with an income so that I could live reasonably comfortably and complete this Doctoral thesis in a timely manner.

To all those people and forces from different spheres of my life journey who have ever talked me down... Your poor words of wisdom fuelled unsurpassed levels of inspiration in me to overcome the deepest depths of self-disbelief that would occasionally infect my mind. I have proven you wrong before, and this is just another milestone you never thought I could attain. Thank you, dearly - I could not have done it without you.

Last, but never least, mum and dad: this one - like all my other successes - is for you! I LOVE YOU!
Chapter 1

Thesis Overview

The mobile agent paradigm presents an elegant and clearly unique method to problem solving in a distributed environment. However, quite a few non-trivial problems are faced by enthusiasts wishing to build and safely utilise mobile agent technology on the Internet. Firstly, there is no ubiquitous framework and standard methodology for designing, building, testing, or deploying mobile agents. Even if there was such a framework, how would potential mobile agent users and mobile agent platform owners - the paradigm’s two major stakeholders - discover and interact with each other in a safe, accountable manner?

The purpose of this thesis is to shed some light on this perplexing question, and to provide a new approach to tackling the security issues associated with the mobile agent paradigm. In this endeavour, this thesis presents an infrastructural strategy devised to enable application and language-independent mobile agents to perform work in a framework whereby both major stakeholders’ security concerns are mitigated without bias in favouring one of the major stakeholder’s interests over the other. Moreover, the infrastructural strategy transparently supports gathering new stakeholders, previously mistrusting by default, in an environment more conducive to interaction and fair outcomes.

Whereas the bulk of prior existing research has focused on mitigating threats
Chapter 1. Thesis Overview

specific to one or the other of the paradigm’s major stakeholders, and even then usually only mitigation for one or two threats, the research perspective of this thesis is formed from a different line of attack. Firstly, this prior style of research on mobile agent security issues is unequivocally identified, and classified as *micro-level* in its approach. The case for the introduction of a wide variety of generic mobile agent infrastructure trusted-third party (TTP) security services is then presented, including how and why they can facilitate both major stakeholders. A novel infrastructural concept, the Mobile Agent Secure Hub Infrastructure (MASHIn), is then introduced with discussion on how some of those generic mobile agent TTP security services can be supported in the MASHIn. Clarification is provided on why the MASHIn approach is a *macro-level* approach to tackling mobile agent security issues, and traditional micro-level mobile agent security issues are considered - specifically mobile agent privacy, execution integrity, and stakeholder reputation - in the context of the novel macro-level MASHIn framework. The devised solutions rate favourably in not only meeting more security properties, but also in providing a means whereby the stakeholders can be transparently brought closer to working relationships.

1.1 Contributions Encapsulated in the Thesis

A number of main contributions presented in this thesis which will be valuable to the mobile agent security research community - and ultimately, it is hoped, mobile agent prospects are identified - including (in no implied order of merit):

1. Account of mobile agent security threats and state-of-the-art countermeasures (including threat mitigation coverage) which is extensive, non-exhaustive, but is an accurate reflection for both mobile agent paradigm major stakeholders.

2. Raised awareness of the importance of a wide application of TTPs in better assisting both mobile agent paradigm major stakeholders.

3. The macro-level MASHIn concept - its strategic utilisation of both traditional distributed system TTPs (registration authorities, certification authorities, and attribute authorities) and novel mobile agent network TTPs (MASHs and checking hosts), as a high-level framework facilitating the deployment of safe and autonomous mobile agents.
4. The *Secure Callback Classes for Mobile Agent Privacy* (SCC4MAP) strategy, and its favourable properties compared with existing mobile agent privacy mechanisms.

5. Definition of new criteria for assessing the robustness of mobile agent execution integrity schemes.

6. The *Real-time Execution Checking and Deterrent Against Misbehaviour* (RECDAM) mobile agent execution integrity scheme, and its high rating compared with existing mobile agent execution integrity mechanisms.

7. A framework for calculating reliable major stakeholder confidence scores - with two major benefits (1) unbiased support for the calculation of personalised, private confidence scores on behalf of both stakeholders, (2) a means of transparently drawing together previously mistrusting stakeholders so that working relationships can be more easily formed.

8. A preliminary investigation and discussion of precautionary measures which should be considered in protecting the two critical MASHIn-specific TTP entities - MASHs and checking hosts.

9. Recommendations for further design and implementing the MASHIn, and presentation of three mobile agent application domain scenarios highlighting the strategic benefits of the MASHIn macro-level approach to promoting safe autonomous mobile agents.

1.2 Thesis Chapter Outline

Organisation of this thesis can be thought of a number of ways; in providing an overview here of the layout of this thesis’ content, let us discuss five main constituent parts:

1. *Non-security-specific introduction to the mobile agent paradigm*: Chapter 2 was intentionally written in a non-security-specific context to give the reader a gentle introduction to the main concepts surrounding the mobile agent paradigm, including a little on its history and factors limiting the ubiquitous uptake of mobile agents on the Internet.
Chapter 1. Thesis Overview

2. Security threats associated with the mobile agent paradigm and state-of-the-art countermeasures: In this part of the thesis, an impression - though non-exhaustive by any means - is made of the sheer volume of threats and non-trivial security issues associated with the mobile agent paradigm, as well as the state-of-the-art countermeasures. Chapter 3 presents a reflective list of some of the major threats to both mobile agent paradigm major stakeholders - mobile agent platform owners, and mobile agent users. Chapter 4 provides a good sample of the state-of-the-art existing strategies for protecting each of the major stakeholders, and ends with a threat mitigation coverage assessment on these existing countermeasures.

3. Highlighting the importance of TTPs in reducing the security issue complexities associated with the mobile agent paradigm: Discussion is commenced in Chapter 5 on how TTPs can be strategically employed to bring significant benefits in terms of reducing security problems associated with the mobile agent paradigm, and importantly lower the mutual inhibitions of both mobile agent paradigm major stakeholders to interact. Chapter 6 provides an introduction to the MASHIn concept including its objectives, planned module abstractions, deployment considerations, credential infrastructure component, and anonimisation and agent location processes.

4. Traditional micro-level mobile agent paradigm security properties addressed in a macro-level context: In this part of the thesis, in-depth discussions on novel strategies for achieving mobile agent privacy, mobile agent execution integrity, and a framework for calculating mobile agent paradigm major stakeholder confidence scores are presented (respectively in Chapter 7, Chapter 8, and Chapter 9). The novel solutions for each of these are designed at a macro-level, in the context of the MASHIn, and their added-value in comparison to traditional micro-level approaches is highlighted.

5. Protecting MASHIn-specific TTPs, implementation recommendations, and conclusions and future work directions: Due to the security importance of MASHIn-specific TTPs, namely MASHs and checking hosts, related fields in distributing computing and operating system security which need to be considered and advanced before the MASHIn concept can be transformed into a viable, practical real-world solution are discussed in Chapter 10. Chapter 11 provides interested parties with guidance on further designing
and implementing the MASHIn concept, as well as presenting a series of three different mobile agent application domain scenarios to highlight the strategic advantage of the MASHIn macro-level approach to mobile agent security. Chapter 12 tails the main body of the thesis with a summary of the main contributions presented in the thesis, conclusions reached, and a discussion on interesting avenues for future work.

Due to the leading major novel contribution of this thesis being the presentation of a macro-level perspective for advancing the prospects of the mobile agent security field, the scope of this thesis is focused at a broad-infrastructure conceptual level. One will not see new low-level cryptographic protocols, details of specific agent system implementations, or running MASHIn applications in this thesis. Such details are either too low-level/application-specific or premature given the high-level conceptual transfer of knowledge intended in producing this thesis.
Chapter 2

Introducing the Mobile Agent Paradigm

Opportunity is missed by most people because it is dressed in overalls and looks like work.

Thomas A. Edison

Mobile agents are an interesting and relatively new technology for performing specific tasks in distributed systems on behalf of their users. The purpose of this chapter is to introduce the reader to mobile agent concepts, as well as to provide a historical perspective for their emergence as an alternative distributed systems processing paradigm.

We start in Section 2.1 by providing a definition for a mobile agent, highlighting which aspects of this sometimes misunderstood abstraction are relevant to us. We provide an introductory description of pertinent mobile agent constituent parts and general concepts related to the mobile agent paradigm in Section 2.2. A mobile agent’s life-cycle is presented in Section 2.3. In Section 2.4 we take a historical look at the emergence of the mobile agent paradigm and its distinguishing processing model in comparison to other distributed system paradigms which were to shape its conception. In Section 2.5 we discuss some application areas which were purported to benefit from the flexibility mobile agent technology offered. Major factors which have limited the widespread take-up of mobile
agent technology in an Internet context are outlined in Section 2.6.

In-depth discussion on security issues associated with the mobile agent paradigm are intentionally omitted from the mobile agent paradigm overview presented in this chapter. They are instead presented in Chapter 3.

### 2.1 Mobile Agents Defined

*Mobile agents* (sometimes referred to as itinerant agents) are software programs that execute and travel through heterogeneous computer networks (usually on behalf of a human user), performing computations and autonomously moving from host to host as required to fulfill goals on behalf of their users [221]. Mobile agents migrate to different hosts to facilitate better interaction with local objects and resources. During this self-initiated migration, a mobile agent carries with it its code and execution state.

Whilst mobile agents do their work on behalf of a human in a computer network, analogous agent entities who work on behalf of other humans or organisations in the concrete ‘real-world’ include secretaries, travel agents, and CIA agents [111].

Many definitions for mobile agents have been given in the literature (just a few can be seen in [72], [75], [111], [173], and [221]). Words common across many mobile agent definitions include ‘program’, ‘autonomous entity’, ‘goal’, ‘computer network’, ‘transport’, ‘data’, ‘code’, ‘processes’, and ‘execution state’. Of interest to us are the following mobile agent characteristics:

- **Itinerant**: The agent must be mobile in that not only can it be transported from its user’s computing environment to a remote host for execution, but also capable of travel to subsequent hosts as required to meet its goals\(^1\). Thus the agent is multi-hop, not single-hop.

- **Encapsulation of a well-defined purpose**: A well-defined task mission has been delegated to the mobile agent from its user.

- **Autonomous behaviour**: The decision to migrate from one host to another rests solely with the mobile agent, excluding further interaction with its

\(^1\)The mobility feature of mobile agents may seem obvious to some, but it is surprising how often agents deployed to a site and stationed solely at that site are incorrectly termed mobile agents.
2.2 Mobile Agent Constituent Parts

user during its mission undertaking\(^2\).

- **Object-oriented structure**: The agent is made up of a set of objects (code logic and attribute components, as well as execution state information), is an active entity, and has an execution thread of control.

*Mobile agent platforms* (sometimes referred to as agent platforms, agent places, agent servers, or agent execution environments) are the execution environments on networked hosts which provide mobile agents an environment for performing computations and capabilities for migrating autonomously to other mobile agent platforms. The heterogeneity of underlying hardware specific to mobile agent platforms and networks connecting mobile agent platforms is hidden from mobile agents via a common language (or set of common languages) in which the program segments of the mobile agents are written.

Although mobile agents are sometimes spoken about in the same vein as Internet worms\(^3\) [76, 168], they differ significantly in that mobile agents usually do not repeatedly clone themselves to multiply their numbers exponentially as malicious propagating worms tend to do throughout the Internet. Moreover, mobile agents rely on a series of common dedicated agent platforms to do work on, usually accumulating partial results as they hop between agent platforms in their mission. Furthermore, agent platforms usually require incoming mobile agents to be authenticated and authorised before allowing them to execute in their environment.

A mobile agent stipulates its travel plans via an *itinerary*. Itineraries may be *fixed* (i.e. list of agent platforms to visit and their order of visitation known in advance of a mobile agent’s mission posting) or *non-fixed* (i.e. any variation on fixed itineraries).

### 2.2 Mobile Agent Constituent Parts

In the most general form, the rudimentary requirements for a mobile agent are:

- *An agent instance ID* used in uniquely identifying and locating a mobile agent in a mobile agent network.

\(^{2}\)Important later in the thesis, we are also particularly interested in ensuring that an agent’s protection is autonomous (i.e. without interaction after initial deployment) of its agent user and home agent platform.

\(^{3}\)The concept of worm programs was first published in 1982 [179]; mobile agents were first conceptualised about a decade later [222].
• *Agent code* which is the software statements instructing the mobile agent to perform some course of actions. This is static for the life-time of the agent.

• *Agent data* can be information defined by the agent’s creator, or acquired or altered via the computation of a mobile agent during the course of its mission. Agent data may be declared static, though generally is dynamic in nature by default (i.e. can be updated).

Mobile agents in some mobile agent systems, for example the Ajanta system [112], also enable an “append-only” mechanism for gathering specific data collected at visited mobile agent platforms in an agent’s itinerary.

Although not required in the general case, additional important constituent parts of mobile agents are static data components like digital certificates (which are very important in security for mobile agents and mobile agent platforms); and an agent specification declaring numerous fields such as who constructed the agent, who the agent user is, and what language the agent is programmed in and is to be interpreted as.

### 2.3 The Life-cycle of a Mobile Agent

Mobile agents are by their very nature dynamic objects. Figure 2.1 depicts a state transition diagram representing the possible states of a typical mobile agent in its life-cycle. A mobile agent starts in the *Created* state when it is newly constructed, or alternatively cloned from an existing mobile agent.

When an action is invoked to start the mobile agent it transitions into the *Executing* state. In this state, a mobile agent performs some calculation or series of instructions, either at its home host or a remote host.

A mobile agent may be automatically or voluntarily transitioned to the *Suspended* state if its execution is suspended, and come back out of this state via an action to resume its execution.

A terminate action, either explicit or by exhaustion of statements to perform, may see the transition of the mobile agent from the execution state to the *Disposed* state and thus the death point in its life-cycle.

When a mobile agent is executing it may seek a local service function of the agent platform, in doing so entering the *Local Service Query* state. The nature of the particular service being sought can vary widely, for example a directory
Figure 2.1: Mobile agent state diagram (adapted from [233]).
services query to locate future agent platforms to travel to or a communications service request to interact with another mobile agent. If the query for a service is granted and utilised the mobile agent returns to either the execution state or if the service granted is to transport the mobile agent to another agent platform the mobile agent will move into the Migrating state. If the service is not granted, the agent platform may either suspend the mobile agent (for example until the service is available) or terminate the mobile agent (in which case the agent transitions to the disposed state).

If the migration attempt is successful, the mobile agent will enter the Arrived at Agent Platform state, otherwise it will enter the halted state. A start action will transition the mobile agent on the new agent platform into the executing state from which the flexible course of the agents dynamic life just outlined may be re-iterated.

2.4 The Evolution of Mobile Agents

Mobile agents remain a relatively new phenomenon, even though they have been furnished and studied since the early-mid 1990s. Research interest in mobile agents was sparked by the possibility of addressing some problems associated with traditionally designed distributed systems, and the emergence of the World Wide Web (WWW) as a viable medium for mainstream society to engage in timely information dissemination and global electronic commerce. These problems included a multiplying increased distributed data repository (i.e. the emerging WWW), the need to find relevant information in a timely manner, the trend towards direct online access to services and goods for large numbers of customers, and the increasing forms of user mobility with intermittent network access and variable device battery lifetimes.

In the following subsections we review the natural evolution path for mobile agents by providing an overview of the preceding distributed system paradigms. In doing so, we highlight their basic modus operandi and additional flexibility in processing tasks in a distributed fashion. We also provide a short discussion on some historically significant mobile agent systems.
2.4. The Evolution of Mobile Agents

2.4.1 Client-Server (CS) Paradigm

The client-server (CS) paradigm is a well known and widely used distributed system paradigm. In this paradigm, a server (for example, site B in Figure 2.2) offers a set of resources to provide some services. The client (site A in our example) requests the execution of a service to the server. Subsequently, site B performs the service requested by executing the corresponding know-how and the result is returned to the client\(^4\) [214].

![Figure 2.2: Client-server paradigm pictured.](image)

The method of request/response just described is the more traditional client-server application. A slightly varied version whereby remote procedures are directly invoked instead are known as Remote Procedure Calls (RPC) applications [195]. RPC increases the flexibility of an architecture by allowing a client component of an application to employ a function call to access a server on a remote system. Both application types fall under the umbrella of the client-server paradigm. This communication model is usually synchronous - meaning the client, after sending the request message to the server, suspends further processing and awaits the results of the call from the server [111, 195].

In client-server applications, the program code (or application logic) is static on the server computer, and the processor (i.e. abstract machine carrying out and maintaining state) and resource (i.e. I/O channels) for computation are supplied by the server.

2.4.2 Remote Evaluation (REV) Paradigm

In the remote evaluation (REV) paradigm, site A (in Figure 2.3) has the know-how necessary to perform a procedure, but not the resources necessary to execute

\(^4\)Actually, a server may rely on other sites in order to perform parts of the required service or to retrieve parts of the required data, but, in this case, the server would act as a client in another client-server interaction. From the original client’s perspective the server owns all the necessary data and knowledge [214].
Chapter 2. Introducing the Mobile Agent Paradigm

the procedure so it instead sends the know-how to another site (site B in our example) which has the necessary resources and is willing to undertake the procedure. The procedure is executed at site B and the result is returned to site A [214].

Figure 2.3: Remote evaluation paradigm pictured.

The main logical difference between the RPC paradigm and the REV paradigm is that in the former (RPC) the client sends the server the name of a procedure, whereas in the later (REV) the body of a procedure is sent [186]. Remote batch processing programs are an example of the remote evaluation paradigm.

2.4.3 Code On Demand (COD) Paradigm

In the code on demand (COD) paradigm, site A (in Figure 2.4) owns the resources needed for the execution of a service, but lacks the know-how needed to use them in performing the service. The corresponding code component can be retrieved from a remote server component (site B), which acts as a code repository. Subsequently, execution takes place on site A.

Figure 2.4: Code on demand paradigm pictured.

This is the scheme typically employed by Java apps [214]. The COD paradigm can be viewed as a form of the REV paradigm, with the client and server roles of the respective entities reversed.
2.4.4 Mobile Agent (MA) Paradigm

The mobile agent (MA) paradigm is a further extension (see Figure\ref{fig:ma-paradigm}) in that not only is the application logic transferred between computers, but the application state can also be transferred from one computer to the next. It is this latter property of mobile agents which facilitates their working autonomously (on behalf of a user) to travel between one or more remote computers, performing some work on those computers as required, often returning some summary or result to the originating home computer.

In contrast from the aforementioned distributed system paradigms, the mobile agent paradigm introduces a new generic concept of an autonomous mobile object. The mobile object encapsulates data along with the operations on that data (as well as some execution context - the mobile object’s state), which can be transported from one site to another in a network under the mobile object’s control (autonomy) as deemed fit by the mobile object (mobile agent) \cite{111}.

Unlike a procedure call, a mobile agent does not usually return a result to the client platform that it was just sent from. Each migration message in the mobile agent paradigm encapsulates mobile object code that the receiving computer is to perform and data that is the mobile object’s contextual supporting argument. This refinement is important and needs to be well understood: code whose execution the sending computer began or continued continues on the receiving computer, with the data constituting the mobile object’s current state \cite{223}.

A mobile agent can choose to migrate to a series of other sites (agent platforms), transmit information back to its origin site, or migrate back to its home agent platform if appropriate (though this is not a requirement of the paradigm). The mobile agent paradigm thus sees an increasing notion of autonomy, enabling significantly more flexibility in a genuinely mobile object over previous distributed system paradigms \cite{111}.

Whilst this autonomous feature enables a more powerful software abstraction, the mobile agent paradigm brings with it some issues of concern - not the least for security of agent platforms and the mobile agents themselves. We will discuss these implications for security in depth, starting in the next chapter.

\footnote{An itinerary of 4 hosts was chosen for reasons of pictorial size only; an arbitrary number of sites to visit in a mobile agent’s itinerary is the more general case.}
2.4.5 Historically Significant Mobile Agent Systems

One of the first systems, though not classified at that time as a mobile agent system, to provide object mobility was the Emerald system [105]. The Emerald system was suitable only in a homogeneous local area networked environment, but otherwise possessed the inherent code mobility foundations which would be the hallmark of mobile agent systems to emerge in the years following it.

Telescript [222], developed in the early 1990s by General Magic, was the first system that popularised the mobile agent paradigm concept and programming mobile agents. The Telescript programming language was an object-oriented language not too dissimilar to C++ and translated program code to an intermediate interpreted form to enable platform portability in a fashion similar to that of Java byte-codes. Although Telescript was ultimately commercially unsuccessful\(^6\) and is no longer available, its historical significance remains undisputed as it will always be an important pioneer system in the field.

Because Telescript was a commercial initiative and to be incorporated in large real-world distributed systems, its security model was an important part of its design [214]. One of the areas that Telescript fared better at than future mobile agent systems based on Java technology was controlling the resource usage of

---

\(^6\) Another proprietary mobile agent system that seems to have failed as a business success story is Concordia by Mitsubishi Electric [137].
2.4. The Evolution of Mobile Agents

mobile agents on agent platforms.

Another advantageous feature of Telescript over those systems was genuine process execution state migration allowing a shipped mobile agent to re-start execution at the instruction immediately following the ship command. In Java-based mobile agent systems, however, an agent must be serialised before migrating, deserialised on arrival at a new agent platform, and always invoked at the same entry point [18].

Agent Tcl [75] and Tacoma [99] were research mobile agent systems constructed shortly after Telescript in the Tcl scripting language and designed principally to work at a low-level on Unix machines. Agent Tcl evolved to support a number of other languages including agents written in Scheme and Java. Tacoma also evolved to support a number of other languages including Python, Scheme, Perl (all interpreted-languages) and even C. In the case of Tacoma mobile agents programmed in C, the code was shipped to the remote site where it was compiled and then executed.

The Aglets mobile agent system [115], developed by an IBM research division in Japan, was significant because it united and widely publicised Java technology with mobile agent development. Today, the majority of agent systems are constructed using Java. An extensive Aglets API was developed and the Aglets project was released as open-source to the community. Around the time Aglets was being marketed, General Magic also released a Java mobile agent system it named Odyssey, but it seems from a lack of mention in the literature that it never received the notoriety of Telescript and it was a shadow of the Aglets mobile agent system in terms of functionality.

There have been a great deal many more mobile agent systems developed, for example in the year 2000 alone a reliable mobile agent system survey had accounted for 72 mobile agent systems [87]. Some of the more prominent mobile agent systems from a security perspective which we touch on later are Ajanta [112], SeMoA [169], SOMA [43], and Gypsy [98].

2.4.6 Mobile Agents Today

As Gray points out [76], many software developers have a range of concerns with utilising mobile agents. On the lower end of these concerns are visualisation, debugging, and maintenance issues. The most pressing concerns relate to security problems with mobile agents which, from an extremist’s view, may put them
only one small step behind the risk of facing viruses and worms if mobile agents are misused. Despite this, Gray and others remain optimistic for mobile agent prospects [76, 101, 173].

Besides widespread ongoing research into mobile agents, currently, there appears to be at least two mobile agent systems being marketed commercially - *Voyager* from Recursion Software [157] and *Jumping Beans* (Inc.) [106], though the market penetration and profitability of these technologies is not clear.

### 2.5 Applications Geared Towards Mobile Agent Technology

The potential benefits of ubiquitous mobile agents are many including dynamic customisation at servers and clients, robust remote interaction over unreliable networks, utility for lightweight client devices and value over low-bandwidth connections, support for disconnected client operations, spontaneous electronic commerce, and facilitation of meaningful information retrieval [36, 60].

Gray [75] suggests that mobile agents are not best viewed as an enabling new technology, but rather as a tool of utility value when implementing distributed applications. Their advantage lies not so much in making new distributed applications possible, but rather in providing a unified programming model and improving the performance of existing applications.

Vigna [214] and Chess et al. [35] point out that the potential application areas that can gain from the strategic employment of mobile agents include distributed information dispersal, distributed information retrieval, advanced telecommunication services, remote device control and configuration, workflow management and cooperation, and electronic commerce.

Mobile agents are a natural evolution of, and complement, previous distributed systems and whilst the individual advantages of agents do not represent an overwhelming motivation for their adoption, the creation of a pervasive agent framework could collectively better facilitate a large number of network-based applications and services [35, 36, 100].

We wish to point out that mobile agents which, in isolation, roam mobile agent networks on their user’s behalf searching only for the best price on an item are a very poor application use of mobile agent technology. These mobile
agents, that we refer to as ‘window shopping mobile agents’, are often used as the hypothetical application when discussing approaches to mobile agent security in the existing body of literature.

If mobile agents are to be seen as an attractive approach to distributed computing it is vital to detail examples which employ the strategic benefits mobile agents can bring, not examples which from an application performance and security perspective are better served by existing client-server technology (such as the ‘window shopping mobile agent’ example).

We will come back to this poor purported application use of mobile agents later in the thesis. In particular, we intend to highlight the security pitfalls of such an application use of mobile agents. Subsequently, we offer an alternative hybrid distributed systems paradigm approach which uses both client-server and mobile agent technology to ensure that security for major stakeholders (i.e. mobile agent users and mobile agent platform owners) and application robustness are of foremost importance. At the same time, we feel our approach does not undermine the value offered by better employment of mobile agent technology.

### 2.6 Factors Limiting the Widespread Adoption of Mobile Agents

Reality has shown that, to date, the incentives offered by mobile agents have not been sufficient to stimulate their widespread deployment despite the fact that the mobile agent paradigm provides an elegant and flexible option for facilitating network services and applications [168].

Although mobile agent technology is being studied for both civil and military applications, one of the major reasons for their limited take-up has been security concerns. Mobile agents not only inherit the security risks associated with traditional distributed system paradigms, but they bring additional new concerns [150].

Vigna recently provided ten reasons, in no particular order of ranking significance, for the failure of mobile agents to live up to their early marketing expectations [215]:

1. Mobile agents do not perform well.

---

7Sometimes, these are also referred to as ‘comparison shopping mobile agents’ in the existing literature.
2. Mobile agents are difficult to design.

3. Mobile agents are difficult to develop.

4. Mobile agents are difficult to test and debug.

5. Mobile agents lack a shared language/ontology.

6. Mobile agents are difficult to authenticate and control.

7. Mobile agents can be brainwashed.

8. Mobile agents cannot keep secrets.

9. Mobile agents are suspiciously similar to Internet worms.

10. Mobile agents lack a ubiquitous infrastructure.

In this thesis, our work helps to partially address problems 6-10 noted above. It is not foolish to suspect that as society becomes more dependent on wireless and mobile computing that greater motivating factors for utilising mobile agent technology will be realised. However, without mechanisms to protect the livelihood of the major mobile agent paradigm stakeholders, interest and investment in the paradigm is likely to remain low.

Johansen [101] makes the point that the research community is largely culpable for the disappointing progress made in developing and utilising mobile agent technology. Despite the fact that those chanting the strategic value of the mobile agent paradigm have developed more than 100 mobile agent systems, the majority, to be fair, hardly provide any incremental contribution to the community as a whole. Yet another Java-based mobile agent system taking advantage of, for instance Java serialisation and reflection support, does not provide much new insight into the fundamental mobile agent challenges - like host and agent integrity, lack of a widely deployed agent infrastructure, fault tolerance, and no common mobile agent standard.

The gap between “accepted” forms of mobile code, such as Java applets, and mobile agents is too large. We cannot expect commercial developers to build mobile agent application systems when security, reliability, and maintenance issues hinder their feasibility as dependable software components [76].

A pervasive Internet security framework for which mobile agent applications can be incorporated is needed [96, 215, 36]. We believe that the prospect of
Internet mobile agents can benefit by taking a higher, macro-level look at security issues concerning the major stakeholders (i.e. mobile agent users and mobile agent platform owners). We propose an infrastructure-oriented approach that mitigates some security problems plaguing mobile agents prospects, in the hope that we can play a small part in lowering the pessimistic response one commonly feels is received on the mentioning of the topic of Internet mobile agents to security-minded computing professionals. We aim to achieve this objective by reducing some of the security concerns faced by the two major mobile agent paradigm stakeholders, in doing so promoting interaction between them.

The focus of the thesis from here on is mobile agent security and our work to reduce some security risks which have limited the widespread adoption of Internet mobile agents.

2.7 Summary

In this chapter we have provided a definition for a mobile agent and the characteristics of this sometimes misunderstood term which are important to us. We have outlined in our introduction the constituent base parts of a typical mobile agent, and we provided an overview of the various states a mobile agent may be in during its life-cycle. We then talked about how the mobile agent paradigm was furnished as a natural extension of previous distributed system paradigms. Although there has been a lot of research community interest in mobile agents over the last decade, there remain areas of significant concern limiting the widespread adoption of Internet mobile agents. Among the major factors limiting commercial interest in deploying mobile agents are security and reliability problems tied to the paradigm, with some of these problems left largely unaddressed to date. Lack of a ubiquitous security framework for safely deploying Internet mobile agents has been another well-cited reason for mobile agents not living up to their original marketing expectations. The remainder of this thesis concentrates on Internet mobile agent security in detail (and to a lesser extent the reliability and robustness of the paradigm). In the next chapter, we commence by discussing in depth the security problems posed by Internet mobile agents. In this process, we will aim to categorise the problems firstly by which major stakeholder (mobile agent user or mobile agent platform owner) a particular problem is most significant to, and then further classify problems appropriately into related risk types.
Chapter 3

Security Problems with the Internet Mobile Agent Paradigm

No one can build his security upon the nobleness of another person.
Willa Cather

In the preceding chapter we introduced the reader to mobile agents and we provided an overview of their emergence as an alternative paradigm to more classical distributed systems based on, for example, the client-server model. We made the point that, despite initial high expectations for their market impact, mobile agents have yet to really take off in an Internet context. One of the major reasons identified for the disappointing adoption of this technology in an Internet environment relates to security concerns for both major stakeholders (i.e. mobile agent users and mobile agent platform owners).

In this chapter we elaborate on security issues pertinent to the mobile agent paradigm and discuss threats specific to each major stakeholder. We start in Section 3.1 by providing an overview of security principles fundamental to protecting more classical distributed systems, like those built on the client-server model, since the mobile agent paradigm not only comprises unique threats specific to its model but also inherits the legacy of threats which are present in more classical distributed systems. In Section 3.2 we look at stakeholders in a mobile agent network and the source and target of perceived threats or attacks. Follow-
Chapter 3. Security Problems with the Internet Mobile Agent Paradigm

In this, we identify and sub-categorise threats specific to each of the two major mobile agent network stakeholders - specifically mobile agent platform owners and mobile agent users in Sections 3.3 and 3.4 respectively.

We do not provide an overview of the state-of-the-art countermeasures for mobile agent security in this chapter. These details are intentionally deferred to the next chapter.

This chapter includes background material found in peer-reviewed accepted papers [i]-[iv] listed on page xxi.

3.1 Fundamental Security Principles of Distributed Systems

In this section we will discuss security principles which are of fundamental importance in protecting all distributed systems, regardless of which specific paradigm the distributed system falls under. We will use the client-server model as the basis for our security principle and threat discussion unless otherwise noted, as all subsequent distributed systems build upon this model of communication and work performed in remote locations to achieve some task.

The subsections below detail security properties which are required to counter specific threats significant in all distributed systems. Where possible, the examples we highlight pertain to messages sent to and from remotely-located distributed system entities.

3.1.1 Authentication

Authentication, in a distributed system, is the process whereby one entity (person or application program) securely establishes to another entity that they are in fact whom they purport themselves to be.

So, for example, consider the scenario in Figure 3.1 whereby Alice wishes to send a message to do some work in collaboration with, or solely utilising, Bob who is located in some remote site.

In the top scenario where there is no authentication involved, even though Alice is acting truthfully in identifying herself, Carol is not acting truthfully and Alice cannot see this as Carol is remotely located and there is no authentication of her message. The solution, in the bottom scenario, is to employ a cryptographic
3.1. Fundamental Security Principles of Distributed Systems

3.1.2 Message Confidentiality

Message confidentiality, in a distributed system, is the process whereby one entity sends a message with the assured security property that only the specific entity for whom it is intended can legibly understand the message - or in other words, an entity securely sends a personal message to another entity.

So, for example, consider the scenario in Figure 3.2 whereby Alice wants to send Bob a message of a personal nature and wishes for Bob (who is remotely located) only to discover the secret of that message - which just happens to be

---

1Authentication schemes can be constructed from both asymmetric (every entity has a unique private and public key) or symmetric (two entities wishing to communicate securely have either a copy of the same key, or a key which can be derived easily from the other’s key) cryptographic schemes, but asymmetric schemes are more common in an Internet messaging context because they scale better.

2It is possible to send a message with the intention for it to be legible only to a subset of recipients, but this requires the employment of a more advanced form of encryption known as group encryption.
that she loves Bob.

![Figure 3.2: Message confidentiality problem/solution in a distributed system.](image)

In the top scenario where there is no message confidentiality, Charles overhears the message from Alice to Bob and the reply message back from Bob to Alice that he, Bob, also loves her. It just happens that Charles was pretty keen on Alice - so, as a result of eavesdropping on the conversation between Alice and Bob, Charles' heart is broken (some might say deserving). The solution, in the bottom scenario, is to employ a cryptographic confidentiality mechanism commonly referred to as message encryption or sometimes message enveloping. Using asymmetric cryptography, for example, one party can encrypt a message using the public key of the intended recipient and only the entity with the corresponding private key (i.e. the intended recipient) can open (or un-envelop) the message and legibly read it. Thus, in the example, even if Charles hears or intercepts either of the encrypted messages from Alice or Bob he will not be able to discern the contents of the message (and he will likely continue to live on in hope for love with Alice).
3.1.3 Message Integrity

Message integrity, in a distributed system, is the process whereby one entity sends a message to another entity with the assured security property that the message can neither be maliciously tampered nor be altered by a transmission error without such a discrepancy being noticed.

So, for example, consider the scenario in Figure 3.3 whereby Alice wishes to send a very clear message to Bob (who is remotely located) and does not want the message - which just happens to be that she loves Bob - to be misconstrued or altered in any way.

In the top scenario where there is no message integrity, Charles intercepts the message from Alice to Bob and alters it to maliciously benefit himself (at the misfortune of both Alice and Bob). He changes the message content to something quite the contrary of the message spirit. Bob is taken aback by the altered message and sends a friendly warning to Alice about Charles which Charles intercepts and alters to benefit himself once more. The solution, in the bottom scenario, that we suggest is to employ a digital signature on an encrypted/enveloped mes-
sage, though using either a digital signature or an encrypted message could have sufficed\(^3\). Employing both in this case means that Charles (and in fact anyone other than Bob) will not be able to read Alice’s message, and Bob will be without doubt that the message is indeed from Alice.

### 3.1.4 Accountability and Message Non-repudiation

*Accountability*, in a distributed system, is the process whereby an entity is held responsible for the actions he/she/it performs in the distributed system. A good example of being held accountable for one’s actions is upholding the non-repudiation property. *Message non-repudiation* is the process whereby an entity sending a message will be held accountable for both the fact the message was sent and the contents of that message.

So, for example, consider the scenario in Figure 3.4 whereby Charles sends a message to Bob (who is remotely located) offering to pay $1,000 to Bob if Bob can tell him the secret to love.

In the top scenario where there is no message non-repudiation, Bob agrees to the deal and responds with his share of the bargain deal (i.e. Bob passes onto Charles his four key qualities to love). Charles gratefully accepts the advice but scoots without paying for the agreed deal. The solution, in the bottom scenario, is to employ digital signatures to the messages forming the sale negotiation dialogue. This ties the entity sending a message to its contents (in this case part of a sale contract). If any disputes arise they can be arbitrated by a third-party or judge at a later time, or a decision can be explicitly bound and enforced by system policy.

### 3.1.5 Anonymity

When communicating with others over the Internet, it is sometimes preferred to not use an identifiable handle (such as a “user name” or any other arbitrary way of identifying who is speaking).

*Anonymity*, in a distributed system, is the process enabling an entity interacting with another entity to remain unaware of who specifically they are communicating with. However, it is often important to know that the communicating

\(^3\)Secure Sockets Layer (SSL) is an example of a technology for ensuring message integrity (and confidentiality) that does not employ a digital signature for every message transmitted in a dialogue between two entities.
Figure 3.4: Message non-repudiation problem/solution in a distributed system.
Chapter 3. Security Problems with the Internet Mobile Agent Paradigm

Entity is a legitimate and authorised entity in the system. One solution to providing this type of anonymity builds on the trusted-third party concept whereby a middle entity keeps the identity of the message sender secret. The recipient target entity is confident that the original message sender is legitimate and authorised to send such a message, yet is unaware of the specific identity of that message sender [34].

3.1.6 Availability and Denial-of-Service

Availability, in a distributed system, is the quality that a remotely located system will be readily available when requested to respond to a service request and the service will be delivered to the client in an acceptable processing time. Thus, the system offering the service must not only be in a capable state on the network for receiving client requests, but also process those requests in a reasonable length of time during peak periods of requests and processing loads.

Denial-of-Service (DoS) is a good example of a threat to the availability of a distributed system server [58]. A denial-of-service attack may be either intentional or accidental in nature. An accidental attack may see a client process or client object with a programming error (like an infinite loop) consuming copious amounts of a server’s processing resources. A robust, well-designed server will normally be well-protected against this type of accidental logic threat. However, an intentional, malicious denial-of-service attack against a distributed system server such as a Distributed-Denial-of-Service (DDoS) attack is much more difficult to defend against. In a DDoS attack multiple remotely-dispersed vulnerable client machines connected to the Internet are hijacked, infected with a malicious program that is initialised to activate at a set time, and used concurrently as the source of repetitive requests to a targeted victim server. Such attacks can reduce the victim server to a crawl because it is unable to respond in a timely manner to the overwhelming number of client requests. In recent years, there have been a number of DDoS attacks against web sites of popular e-commerce vendors such as eBay and Amazon, computing vendors and portals like Microsoft and Yahoo, and government bodies and political news agencies like the United States Whitehouse and Al-Jazeera which have taken these sites off the Internet for at least a couple of hours at a time. Even the set of thirteen Internet root DNS servers have been targeted by dedicated DDoS attacks [54, 127, 28].
3.2 Avenues of Threats in a Mobile Agent Network

Failure to adhere to fundamental distributed system security properties in a mobile agent network context leads to an increased susceptibility to threats, including the ones outlined below. The source and target entities of security threats in a mobile agent network are identified in this section.

Jansen and Karygiannis [96] point out that, though a number of code mobility and mobile agent framework models exist, for focusing on mobile agent security issues it is sufficient for one to picture a simple yet accurate model involving only two main components - the agent and the agent platform as depicted in Figure 3.5.

![Figure 3.5: Fundamental agent system model (adapted from [96]).](image)

From the model in Figure 3.5 it is possible to identify four threat categories in a mobile agent network, specifically threats stemming from [96]:

1. An agent attacking an agent platform: Threats in which agents exploit security weaknesses of an agent platform or launch attacks against an agent platform. Denial-of-service and unauthorised access are among the list of
threats in this category. For an in-depth elaboration of threats to an agent platform from mobile agents see Section 3.3.

2. An agent platform attacking an agent: Threats in which agent platforms compromise the security of mobile agents. This set of threats includes masquerading as another agent platform or trusted third party, denial-of-service, eavesdropping, and alteration of one or more of the mobile agent code/state/data properties. For an in-depth elaboration of threats to a mobile agent from agent platforms see Section 3.4.

3. An agent attacking another agent on the agent platform: Threats in which agents exploit security weaknesses of other agents or launch attacks against other agents. This set of threats includes masquerading as another agent, unauthorised access, denial-of-service, and repudiation of an inter-agent transaction. Many agent platform components may be implemented as agents themselves, though they are usually stationary agents. In any case, countering this set of threats is largely dependent on agent platform system-level precautionary measures. Vulnerability to this threat set is also dependent on how the agent is programmed - specifically which types of inter-agent communication are permitted, and the context and extent of this inter-agent communication.

4. Other entities attacking the agent system: Threats in which external entities, including agents and agent platforms, threaten the security of the agent system framework, usually by attacking agent platforms or inter-platform communications (and thus also indirectly mobile agents). This set of threats includes masquerading, denial-of-service, unauthorised access, and copy and replay. Once more, countering this set of threats is heavily dependent on agent platform precautionary measures.

Fundamentally requisite to countering the set of threats in categories 3 and 4 above are agent platform precautionary measures necessary to counter threats from category 1. In this thesis, we therefore only concentrate on minimising threat concerns from categories 1 and 2 detailed above (see Sections 3.3 and 3.4 respectively for a more detailed threat discussion).

Without agent platforms to do work on, mobile agents are a moot proposition. Likewise, without an influx of mobile agents to service, agent platforms are a moot proposition. As we have shown, the general model of a mobile agent system
consists of only two main stakeholder entities - mobile agents and agent platforms. The majority of threat sources in a mobile agent system are either from mobile agents or agent platforms, with the target of the threats being the other major stakeholder. There, thus, exists an environment of mutual mistrust and fear between the two major stakeholders in a mobile agent system. Neither stakeholder wants their first or next step in an interaction with the other to be a losing one. This state of apprehension between the two major stakeholders is a serious complication limiting the advancement of Internet mobile agent prospects.

We feel that broadening the mobile agent model to incorporate trusted third parties can assist in mitigating some threats encompassed in the four categories above, as well as lowering the lingering mistrust between the two major mobile agent network stakeholders. We see this high-level strategy as a novel, important approach to the advancement of the mobile agent field.

### 3.3 Threats to Mobile Agent Platform Owners

The following subsections discuss some leading threats to mobile agent platforms and, thus also, to the business continuity and reputation of administrators/owners in providing safe and reliable agent platforms. They are an extension of the threats faced when a user downloads an untrusted Java applet to be interpreted on their local machine (i.e. an extension of threats from applications based on the code on demand paradigm).

#### 3.3.1 Denial-of-Service from Agents

As highlighted earlier in Section 3.1.6, denial-of-service attacks are a serious threat to all networked systems. In the context of mobile agent systems, a mobile agent granted permission to execute on an agent platform can launch denial-of-service attacks by consuming an excessive proportion of the agent platform’s computing resources (such as disk space, CPU load, open network ports, or system-allocated file handles [111]) - either intentionally by running attack scripts to exploit system vulnerabilities or unintentionally via programming errors. These attacks can be so severe that they completely shut down or cripple the agent platform in its effort to service other agents [96].

As the majority of today’s agent platforms rely on Java’s security model for host protection, exploiting weaknesses in the Java security model is an obvious
avenue of interest for those persons wishing to attack agent platforms. Among many safety and security concerns with the standard edition of Java include some pointed out by Roth [168]:

- Lack of resource control, with hoped-for extensions not expected in the near future;
- Lack of application separation, not to be addressed until at least JDK version 1.6 (current version is 1.5);
- No safe method to force a Java thread to stop, with hostile code able to catch any exceptions pertaining to its elimination;
- The garbage collector thread may be hijacked by either directly or indirectly overriding finalisation methods;
- Malicious code may block on globally visible class locks, thereby locking other threads vital to the functioning of the runtime system.

An impressive number of security holes have been detected in implementations of Sun Microsystems’ Java Virtual Machine (JVM). The typical response from Sun Microsystems has been that they relate to implementation specific flaws, and no fault could be attributed to Java’s security model. Whilst such an attitude and risk with respect to system security might be acceptable in low-risk domains, it is intolerable in commercial environments for which Internet mobile agents would be applied [154].

### 3.3.2 Mobile Agent Gains Unauthorised Access

Unauthorised access may occur simply through a lack of adequate access control mechanisms at the agent platform or weak identification and authentication, which allows a mobile agent to masquerade as one trusted by the platform. Once (unauthorised) access is gained, information residing at the platform can be disclosed or altered and services the mobile agent usually should not have privilege to use are used without such authorisation. Besides confidential data, this information could include the invocation of system-level instruction codes on the platform [94].

A safe binding between the visiting agent code and the local environment (including system resources and other agents) must enable a mobile agent to perform
its authorised functions without compromising system security and avoiding the malevolent disturbance of other agents [111, 96].

### 3.3.3 Agent Refusal to Pay for Services Delivered to It

A mobile agent may repudiate having received, and thus refuse to pay for, services delivered to it by an agent platform. Repudiation often occurs within non-agent systems and real-life business transactions within an organisation - for example documents are: occasionally forged, often lost, created by someone without authorisation, or modified without being properly reviewed [96]. Because the agent platform’s processor is in control of the agent whilst executing the agent on the agent platform there is a significant and legitimate case for subsequent repudiation by an agent or agent owner (whether that repudiation is based on malicious or mistaken pretentions). Such an attack can lead to serious disputes, litigation, and financial losses regardless of whether the repudiation is well-founded.

### 3.3.4 Infection via Malicious Agent Software Forms

Mobile agents are components that autonomously trigger the transfer of their image to a remote agent platform where they restart execution. This mechanism has much similarity to the method in which malicious worms spread across networked systems. Internet worms have proven to be extremely difficult to eradicate. A mobile agent infrastructure could by default unknowingly support the execution of both benign and malicious agents [215].

Mobile agents are not the only method by which viruses, worms, or other forms of active malicious software such as trojan horses might be propagated in networks, though the use of mobile agents may greatly facilitate such propagation [36].

Authentication and digital signatures cannot, *per se*, guarantee that a particular mobile agent is harmless. Cryptography suffers from a principal problem when applied to agent scenarios: an agent platform might trust the agent’s creator, but how does the agent platform know that an intermediate host did not tamper with the agent? In large-scale mobile agent networks, the trust-basis is problematic since the more entities (systems or agents) an entity trusts, the greater the accumulative risk that entity is invariably exposed to - both directly and indirectly [154].
The active malicious components in an agent can be introduced either intentionally (by a malicious agent owner or intermediate agent platform owner) or accidentally (by a virus infection, or logic flaw in agent or middleware design).

### 3.3.5 Attacks Against Inter-agent or Platform Communications

Other entities (besides agents) may target agent platforms by attacking the inter-agent and inter-platform communication channels through masquerade (e.g. via forgery or replay) or intercept. One possible attack, at a protocol level below the agent-to-agent or platform-to-platform communication layer, may enable eavesdropping or manipulation of messages (perhaps introducing viruses or trojan horses) in transit to and from a target mobile agent or mobile agent platform [94, 96].

As a mobile agent network and the number of involved entities grows larger, so too increases the possibility for successful attacks. For example, a state conducive to on-demand secure inter-messaging may not always be feasible with new (previously unknown) entities who do not have, or share, compatible publicly-verifiable cryptographic keying material.

### 3.4 Threats to Mobile Agent Users

The following subsections discuss some leading threats to mobile agents and, thus also, to the confidence a mobile agent user can place in the delegation of work to, and results of, a mobile agent launched onto the Internet. These threats are by far (in terms of both number and severity) the more non-trivial set of attacks to defend against in mobile agent systems. Traditional distributed system attack countermeasures were not devised to address threats stemming from attacks on the application by the execution environment, which is precisely the situation faced by a mobile agent executing on an untrusted agent platform.

#### 3.4.1 Impersonation from an Interacting Agent or Agent Platform

An agent or an agent platform may masquerade as another one by faking the identity of the legitimate agent or agent platform. This could lead the agent
to be fooled into trusting them with data (e.g. an adversary agent asks agents for their agent user’s telephone number, address, and date of birth) and critical resources (e.g. trusted execution of a sensitive agent by a masquerading agent platform) that they would not ordinarily be privileged to [233]. An agent platform masquerading as a trusted third party may be able to lure unsuspecting agents to the agent platform and extract sensitive information from these agents. A masquerading agent platform can do more harm than a masquerading agent as the latter can harm other agents only through the messages they exchange and/or an agent’s actions resulting indirectly from this exchange, whereas a malicious masquerading agent platform can subvert the agent directly to many additional threats [96], including those mentioned below.

### 3.4.2 Agent Platform Denies Proper Service to Agent

Given an agent has permission to travel to an agent platform, the mobile agent user expects the agent platform to execute the agent’s requests faithfully, provide fair allocation of resources, and abide by any existing quality of service agreements. A maliciously-acting agent platform, however, may ignore agent service requests, terminate the agent prematurely without notification, or simply not execute the agent’s code at all. The malicious agent platform may also introduce unacceptable delays for time-critical agent tasks (such as placing orders to purchase certain stocks in a stock market, or purchasing from a limited-number airline tickets advertised at a special price), or pretend it does not know of resources available to it (such as Java libraries). These attacks could cause “deadlock” where an agent on another agent platform cannot proceed as it is awaiting results from the non-responsive agent on the attacking agent platform. “Livelock” is another possibility; a livelocked agent differs conversely from a deadlocked agent, in that it is not blocked or waiting for anything, but is continuously given tasks inappropriately by an attacking agent platform to perform and can never catch up or achieve its goal [96].

### 3.4.3 Agent Platform Obtains Private Data or Mines the Agent

An agent platform may misuse its privileges over the operating system to obtain the private data of an agent, thus leaking private agent data. Agent platforms
“see” the agent’s code as they execute it, though it may be possible to see the code even before it is executed (and certainly after it is executed if it is recorded). The agent platform may analyse the agent code to learn more about the agent’s execution strategy. Such an attack leaves no discernible evidence for the mobile agent user [233]. Personal information such as email addresses and agent itinerary data may be mined by malicious agent platforms and form the basis of future spamming attacks. Even though the agent may not be directly exposing secret information, the agent platform may be able to infer meaning from the types of services requested or messages passed to other agents. For example, someone’s agent may be communicating with a travel agent. Although the content of the message may not be exposed, this communication may indicate that the agent user is planning a trip and will be away from their home in a given period of time. The agent platform may non-consentingly share this information with a suitcase manufacturer who sends advertisements to target users, or even worse, the agent platform administrators may share this information with thieves who may target the home of the traveller [96].

### 3.4.4 Eavesdropping of Agent Messages

Attacks here are similar to the classical third-party eavesdropping threats (as explained in Section 3.1.2), except much harder to defend against if the attacking third-party is the agent platform on which the agent is sending or receiving the message. The agent platforms can pry on inter-agent communications, even if secure messaging is employed because the encrypting and decrypting key - or symmetric session messaging key - is visible to the interpreting host agent platform [96, 60].

### 3.4.5 Manipulation of Inter-agent Messages

Like the previous threat, defending against an attack from a mobile agent platform that is altering intended inter-agent messages is significantly more difficult than defending against an attack from the classical inserted adversary in a distributed system message integrity attack (as explained in Section 3.1.3). If an agent platform decides to behave malevolently, it may wrongly generate inter-agent messages, not send or deliver inter-agent messages as expected, or alter the contents of inter-agent messages because it has control over how the agent
3.4. Threats to Mobile Agent Users

is interpreted. For example, a message to purchase some item by the agent may be altered to increase the purchase quantity, or replayed continuously until the agent (or its user) is bankrupt.

3.4.6 Agent Platform Modifies the Agent’s Code or State

If an agent platform is able to read the code and if it has access to the code memory of an agent (the usual case), it can modify the program (code and/or state) of an agent. It could exploit its role by altering the code permanently (e.g. implanting a virus, worm, or trojan horse). Alternatively, the agent platform could temporarily alter the behaviour of the agent on its platform only (and forward unmodified code in the agent to the next agent platform in the agent’s itinerary). The advantage of the latter approach is that the agent platform to which the agent next migrates will not be able to detect manipulation of the code since the version it received has not been modified from the version the attacking previous agent platform received [84].

3.4.7 Agent Platform Manipulates Agent’s Control Flow

Even if the agent platform does not have clear-text access to the data of the agent (for example a competitor may have encrypted some input data using the public key of the mobile agent user and appropriately updated the agent), an attacking agent platform can misuse its responsibility to interpret the agent’s behaviour as it is provided by manipulating the agent’s control flow [84]. From this threat, an attack could see the agent platform skip all executed statements and forward the agent onto the next agent platform in the agent’s itinerary. In another situation, the agent platform may choose an alternative course to manipulating the control flow of the agent like jumping to the logical step when some threshold has (apparently) been reached. This threshold might be the state when an item bargain price has been found, so buy the desired item. Thus, a malicious agent platform can coerce the agent into performing some action it would not ordinarily do given genuine inputs from the agent platform.

3.4.8 Agent Platform Misroutes the Agent

A simple attack by an agent platform would see the agent sent to an agent platform not designated by the agent (under its autonomous discretion) as the
next intended agent platform to visit in its travels. This threat also implies that
the agent platform may attack by maliciously sending a copy of the agent to a
non-intended recipient (agent platform or some other entity) as well as sending
the agent to its intended next agent platform. This latter variety of attack is
much more difficult to detect and counter.

3.4.9  Replay Execution Attack on an Agent

With each visit to a mobile agent platform, a mobile agent increases its expo-
sure to security threats (and thus attacks). An agent platform may re-execute
an agent (potentially unlimited times) to see if it can gather a secret (possibly a
hidden secret key) or learn more about the agent’s intentions. When readied for
migration, the agent transported to the next agent platform in its itinerary would
be the agent data state that was output from the first (and only legitimate) exe-
cution of the agent by the maliciously probing agent platform, therefore passing
on no evidence that a replay execution attack took place.

3.4.10  Agent Platform Repudiation of Communication

A malicious agent platform can deny having conducted a communication ex-
change with an agent, for example quoting the vendor’s preferred price on an
item. A malicious agent platform can also deny having received payment from
the agent for delivered services to, or purchases made by, the agent when in fact
the payment transaction did take place.

3.4.11  Agent Platform Repudiation of Execution

A malicious agent platform can deny having executed an agent when indeed it
did. Furthermore, a malicious mobile agent platform, upon a real-time query
from an external entity, can deny that it is executing a mobile agent when indeed
it is.

3.4.12  Agent Platform Repudiation of Agent Migration

A malicious agent platform may deny having received or sent an agent from/to
an agent platform. Such a denial may be made in an attempt to free itself from
culpability for any damage imparted on a mobile agent.
3.4.13 Agent Platform Returns Wrong Results

An agent platform running a database server may manipulate the information obtained from the database, or modify the database directly, therefore the mobile agent will obtain false information [33]. The results of system calls such as date or time information may also be incorrectly returned by the operating agent platform environment to the mobile agent [84].

3.5 Summary

In this chapter we have studied threats to the Internet mobile agent paradigm, and specifically to its two major stakeholders - mobile agent platform owners and mobile agent users. Given that the mobile agent paradigm is an extension of its ancestor distributed system paradigms (as discussed in the previous chapter, Section 2.4), we firstly presented an overview of the security principles which are fundamental in guarding against classical distributed system attacks. These distributed system security principles form the starting point for understanding and countering mobile agent system threats, though as we explained the Internet mobile agent paradigm brings with it a large range of new threats that traditional distributed system protection mechanisms were not devised to guard against. One such set of attacks are those arising from a malicious remote executing environment on code deployed to execute and work autonomously in its entirety remotely on behalf of an end-user. This is exactly the scenario an attacking agent platform exploits when it malevolently interprets an agent.

Table 3.1 lists the threats we discussed in this chapter, notes the specified major stakeholder targeted in the attack, provides a general-case rating for how difficult the threat is to mitigate, and notes whether our proposed infrastructure-oriented mobile agent security strategy (i.e. the thesis contribution) can provide a solution to thwart the threat.

In the next chapter we will provide a review of the state-of-the-art countermeasures available to mitigate the threats identified in this chapter. In conducting this review, we will be further able to highlight the particular difficulties in countering some threats more than others. As we will show, while many countermeasures have been proposed they often suffer from significant limitations in their practicality or robustness, and their lack of interoperability is also of concern. Perhaps the biggest hurdle to the advancement of the Internet mobile agent
Chapter 3. Security Problems with the Internet Mobile Agent Paradigm

<table>
<thead>
<tr>
<th>Threat</th>
<th>Target of Threat</th>
<th>Feasibility to Mitigate Threat</th>
<th>We Offer a Solution?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denial-of-Service from Agents</td>
<td>APO</td>
<td>Difficult</td>
<td>Partially</td>
</tr>
<tr>
<td>Mobile Agent Gains Unauthorised Access</td>
<td>APO</td>
<td>Easy</td>
<td>Partially</td>
</tr>
<tr>
<td>Agent Refusal to Pay for Services Delivered to It</td>
<td>APO</td>
<td>Moderate</td>
<td>Partially</td>
</tr>
<tr>
<td>Infection via Malicious Agent Software Forms</td>
<td>APO</td>
<td>Moderate</td>
<td>Partially</td>
</tr>
<tr>
<td>Attacks Against Inter-agent or Platform Communications</td>
<td>APO</td>
<td>Easy</td>
<td>Yes</td>
</tr>
<tr>
<td>Impersonation from an Interacting Agent or Agent Platform</td>
<td>AU</td>
<td>Moderate</td>
<td>Partially</td>
</tr>
<tr>
<td>Agent Platform Denies Proper Service to Agent</td>
<td>AU</td>
<td>Difficult</td>
<td>Yes</td>
</tr>
<tr>
<td>Agent Platform Obtains Private Data or Mines the Agent</td>
<td>AU</td>
<td>Difficult</td>
<td>Yes</td>
</tr>
<tr>
<td>Eavesdropping of Agent Messages</td>
<td>AU</td>
<td>Difficult</td>
<td>Yes</td>
</tr>
<tr>
<td>Manipulation of Inter-agent Messages</td>
<td>AU</td>
<td>Difficult</td>
<td>Yes</td>
</tr>
<tr>
<td>Agent Platform Modifies the Agents Code or State</td>
<td>AU</td>
<td>Difficult</td>
<td>Partially</td>
</tr>
<tr>
<td>Agent Platform Manipulates Agents Control Flow</td>
<td>AU</td>
<td>Difficult</td>
<td>Partially</td>
</tr>
<tr>
<td>Agent Platform Misroutes the Agent</td>
<td>AU</td>
<td>Moderate</td>
<td>Partially</td>
</tr>
<tr>
<td>Replay Execution Attack on an Agent</td>
<td>AU</td>
<td>Difficult</td>
<td>Partially</td>
</tr>
<tr>
<td>Agent Platform Repudiation of Communication</td>
<td>AU</td>
<td>Moderate</td>
<td>Partially</td>
</tr>
<tr>
<td>Agent Platform Repudiation of Execution</td>
<td>AU</td>
<td>Moderate</td>
<td>Yes</td>
</tr>
<tr>
<td>Agent Platform Repudiation of Agent Migration</td>
<td>AU</td>
<td>Easy</td>
<td>Yes</td>
</tr>
<tr>
<td>Agent Platform Returns Wrong Results</td>
<td>AU</td>
<td>Difficult</td>
<td>Partially</td>
</tr>
</tbody>
</table>

Table Key: APO = Agent Platform Owner, AU = Agent User

Table 3.1: Summary of noted threats, general-case mitigation feasibility, and whether our proposal offers a solution to thwarting the threat.

paradigm that researchers have not touched on in any great depth is the (almost too obvious) fact that if the two mobile agent system stakeholders (i.e. mobile agent platform owners and mobile agent users) mutually mistrust and avoid each other because they see the threats outweighing the benefits of interaction, there is no chance of real-world progress in this field because both stakeholders rely on the other for the mobile agent concept to proceed! Our intention in later chapters is thus not only to offer mitigation to a number of threats facing the two major stakeholders individually, but to provide a generic high-level approach which enables the two sets of major stakeholders to come closer together and establish a working relationship.
Chapter 4

Existing Strategies for Protecting Major Mobile Agent System Stakeholders

Glory built on selfish principles is shame and guilt.

William Cowper

In the previous chapter we highlighted a number of significant security issues tied to the Internet mobile agent system model. We identified and presented an overview of a large number of security threats targeted at the two major mobile agent system model stakeholders (i.e. mobile agent users and mobile agent platform owners). We noted that if these stakeholders are mutually mistrusting to the extent that they do not wish to engage in interaction because they fear their next step may be a losing one, there will be no practical advancement of the Internet mobile agent paradigm.

This chapter is a review of the state-of-the-art approaches found in the existing body of literature to protect the major mobile agent system stakeholders. We do not claim it is an exhaustive list of approaches, but rather a good representation of the commonly referenced approaches. Where possible, we will organise these approaches in an order of symmetry to the threats presented in the previous chapter, though because some approaches aim to address more than one
Chapter 4. Existing Strategies for Protecting Major Mobile Agent System Stakeholders

This chapter includes background material found in peer-reviewed accepted papers [i]-[iv] listed on page xxi.

4.1 Countermeasures for Thwarting Threats to Mobile Agent Platforms

Many techniques for protecting mobile agent platforms from malicious or erroneous mobile agents have been proposed and implemented. They can be correlated to protection mechanisms associated with downloaded executable content in general with the added complexity that a mobile agent, arriving at any specific agent platform in its itinerary, may have already visited agent platforms that the hosting platform does not trust. A list of approaches for countering the risks to hosting agent platforms from mobile agents will be discussed in the ensuing subsections.

4.1.1 Software-based Fault Isolation and Sandboxing

Software-based fault isolation is a programming language independent approach to implementing fault isolation within a single memory address space. A prototype applying the principles of the software-based fault isolation mechanism was built in a RPC environment [217], but could be extended to the mobile agent environment. Distributed code and data modules are separated, managed by software, into disjoint fault domains (i.e. logically separate portions of application address space). The object code is then modified so that it cannot jump or
write outside its assigned own fault domain. The software encapsulation technique of sandboxing, later employed and popularised by the Java security model, is used to prevent distributed modules from escaping their own fault domain. The sandboxing mechanism, by itself, is somewhat of an inflexible mechanism since it suffers from an all or nothing approach. This limitation can be alleviated to a certain degree if code signing (discussed in Section 4.1.4) is utilised also. Leading Java-based mobile agent systems all rely on JVM sandboxing as a foundation in protecting agent platforms [23, 169, 112, 43, 115].

The concept of disjoint fault domains and localised damage containment on attempted penetrations from malicious code is also an underpinning of the mandatory access and jailed process abstractions designed into the Security-Enhanced Linux (SELinux) architecture [120]. The SELinux kernel enforces mandatory access control policies that confine user programs and system servers to the minimum amount of privilege they require to do their jobs (with no equivalent "root"/super-user). When confined in this way, the ability of these user programs and system daemons to cause harm when compromised (via buffer overflows or mis-configurations, for example) is reduced or eliminated. This confinement mechanism operates independently of the traditional Linux access control mechanisms [139].

4.1.2 Safe Code Interpretation and Using Type Safe Languages

The strategy of safe code interpretation is such that commands considered harmful are either made safe for, or denied to, a mobile agent. As noted already in Section 2.4.5 many agent systems have been programmed using the Java language, and rely on strong sandboxing policy enforcement. Java Virtual Machines (JVMs) are designed to restrict memory and method access, and maintain mutually exclusive execution domains. Type safety is the means by which the compiler ensures that methods and programs do not access memory in ways that are considered dangerous; it is a crucial part of the Java security model since it fundamentally protects the integrity of the memory map by disallowing unprivileged code to access protected parts of memory [94, 189].

Some agent systems seem to rely largely (and sometimes solely) on core Java system security measures for protecting agent platforms [23]. Such design is inherently non-robust since, as highlighted in Section 3.3.1, the Java Virtual
Machine environment has a number of design weaknesses which can be readily exploited [168].

4.1.3 Proof-carrying Code

Proof-carrying code is a software mechanism based on well-known principles from logic, type theory and formal verification that allows an agent platform (i.e. the code consumer) to determine with certainty that it is safe to execute a program supplied by a untrusted mobile agent (i.e. the code producer) [140]. The aim of the proof-carrying code (PCC) approach is to prevent the execution of unsafe code by sending the code and proof together to the code consumer where the safety properties can be verified. A safety predicate, representing the semantics of the program, is generated directly from the native code to ensure the companion proof does in fact correspond to the code. The proof is structured in a way that makes it straightforward to verify without using cryptographic techniques or external assistance. Once verified, the code can run without further checking. Any attempts to tamper with either the code or the safety proof result in either a verification error or, if the verification succeeds, a safe code transformation.

By strategically employing the PCC mechanism, safety policies can be defined that stipulate not only standard requirements such as memory safety, but also more abstract and fine-grained guarantees about the integrity of data abstraction boundaries. In this respect, PCC goes beyond the safeguards provided by some host protection mechanisms such as software-based fault isolation (discussed in Section 4.1.1) [233, 140].

There are, however, some potentially difficult problems to solve before the PCC approach can be considered widely practical. They include a standard formalism for establishing security policy, automated assistance for the generation of proofs, and techniques for limiting the potentially large size of proofs that could arise. Furthermore, the technique is semi-dependent on hardware and operating environment of the code consumer (i.e. mobile agent platform), which may limit its widespread applicability [94].

4.1.4 Code Signing

A fundamental technique for protecting an agent platform involves the application of a digital signature on some incoming mobile agent code or other object. The
4.1. Countermeasures for Thwarting Threats to Mobile Agent Platforms

primary goal is to establish accountability by authenticating the principal that lends a mobile agent its authority and vouches for its correctness [233].

Code signing, building on earlier research to contain computer viruses [155], also protects the integrity of transmitted code using cryptographic hashing. Moreover, it can be used to ensure the code has been certified by a principal - the code signer - via the application of a cryptographically-generated digital signature. The signing principal may be the creator of the mobile agent, the user of the mobile agent, or some other entity that credits the mobile agent’s code as safe. More than one of these principals may apply their digital signature to the code, vouching their approval of the code. Because a mobile agent operates on behalf of an end-user, mobile agent systems commonly use the signature of the end-user as an indication of the authority under which the mobile agent operates [233, 96].

In (second-generation) Java, the code signing [190] of remote applets or mobile agents can allow code from reputable entities fine-grained access to local resources when combined with domain-specific security policies [191], adding much greater flexibility to the sandboxing directive (discussed in Section 4.1.1). Microsoft’s Authenticode [130], a common form of code signing, enables Java applets or Active X controls to be signed, ensuring users that the software has not been tampered with or modified and that the identity of the author is verified.

Too often, however, code signing is thought of guaranteeing more than it really does [154]. Code signing does not eliminate the problem of malicious or erroneous code (because a digital signature applied on software code is no guarantee that obvious or subtle vulnerabilities are not embedded in the code), and thus should not be seen, by itself, to be such a claim. Jansen and Karygiannis [96] suggest that rather than relying solely on the reputation of a code producer, one can gain greater assurance from an independent review and verification of code performed by a trusted party or rating service. IBM’s FlexxGuard [91] is a software product that dynamically derives protection domains for incoming distributed objects and can be utilised within a code rating infrastructure. Nevertheless, strict enforcement of access control policies should never be bypassed simply on the favourable rating of remote code by any third party.

4.1.5 State Appraisal

State appraisal aims to show that a mobile agent (in a multi-hop itinerary) has not been subverted due to alterations of its state information [59]. Both the
author and user of a mobile agent produce appraisal functions that become part of a mobile agent’s code. These are later used to determine the privileges granted to a mobile agent based on conditional factors and whether the state invariants hold, in combination with an agent platform’s security policy. Thus, a mobile agent whose state violates an invariant can be granted no privileges, while a mobile agent whose state fails to meet some conditional factors may be granted a restricted set of privileges.

The success of the state appraisal technique for agent platform protection relies on the extent to which harmful alterations to a mobile agent’s state can be predicted, and countermeasures in the form of appraisal functions that can be prepared before invoking the mobile agent. As the state appraisal designers note, not all malicious state alterations can be detected because agents travel to new hosts to acquire information that is not available elsewhere. This may allow some deceptive alterations to be indistinguishable from the normal results of different (but possible) information on a remote host [59]. Moreover, the state space for a mobile agent could be quite large and with each new mobile agent application scenario comes a new set of possible states and conditions - suggesting significant scalability and practicality limitations with this proposed mechanism [94].

4.1.6 Path Histories

Path histories maintain an authenticable record of the prior platforms visited by a mobile agent, so that a newly visited platform can determine whether to process the mobile agent and what resource constraints to apply [152, 35]. Computing a path history requires each agent platform to add a signed entry to the path, indicating its identity and the identity of the next platform to be visited, and to supply the complete path history to the next platform. A platform can determine whether it trusts the previous agent platforms that the mobile agent visited, either by simply reviewing the list of identities provided or by individually authenticating the signatures of each entry in the path history to confirm identities.

The technique does not prevent a platform from behaving maliciously on the mobile agent code/data/state, and collaboration attacks between successive agent platforms cannot be detected. As the path history, or number of hops a mobile agent takes in its itinerary, increases so does the time to compute path verification [94]. Furthermore, the greater the path history, the greater the likelihood
that an agent platform will not trust the mobile agent because it has visited an agent platform the agent platform mistrusts or knows nothing about.

4.1.7 Summary of Measures to Protect Agent Platforms

Conventional security techniques, such as code signing (discussed in Section 4.1.4), commonly used in contemporary distributed systems (e.g. those based on the client-server model) can often be utilised in securing agent platforms in the mobile agent paradigm. There are a number of extensions to conventional mechanisms and techniques devised specifically for controlling mobile code and executable content (e.g. Java applets) that are applicable to mobile agent security [96] - for example sandboxing (discussed in Section 4.1.1), and safe code interpretation and using type safe languages (discussed in Section 4.1.2). There have also been mobile agent specific countermeasures devised which aim to protect agent platforms against various threats, such as state appraisal (discussed in Section 4.1.5) and path histories (discussed in Section 4.1.6). Of the two major stakeholders, protection of agent platform owners (and their agent platforms) is by far the more well-studied and developed area in terms of viable and effective techniques. In the next section we look at protection measures to aid mobile agents in thwarting attacks directed at them (usually from malicious agent platforms).

4.2 Countermeasures for Thwarting Threats to Mobile Agents

By far the more non-trivial set of attacks to defend against in mobile agent systems are those arising from malicious hosting agent platforms. Traditional distributed system attack countermeasures were not devised to address threats stemming from attacks on the application by the execution environment, which is precisely the situation faced by a mobile agent executing on an untrusted agent platform [94].

Since mobile agents are executed on agent platforms, each instruction in the mobile agent’s code is observed by the controlling machine or virtual machine which also maintains the mobile agent’s state. This causes a number of security concerns of which the most prominent ones are: the integrity of the mobile agent, in particular the integrity of its mutable part, needs to be protected; maintaining
Chapter 4. Existing Strategies for Protecting Major Mobile Agent System Stakeholders

the secrecy of the mobile agent’s computations and data is a fundamental requirement for fair negotiations as well as for computations on confidential information such as the preferences (or profile) of the mobile agent’s owner, or secret keys; and, protecting the integrity of the mobile agent’s control flow is a precondition for any mobile agent to trust its own decisions [164].

In the following review of the state-of-the-art countermeasure proposals to the threats posed by the malicious host platform problem, we have attempted to break up the review into logically related countermeasure approach groupings which are geared towards assisting the thesis’s major research and investigation objectives (as outlined in Section 1).

4.2.1 Approaches Which Are Unsuitable

There are a number of suggested approaches for mitigating the pressing mobile agent security risks associated with the malicious host platform which are simply infeasible in a high-level framework solution that we aim to deliver. We review only a couple of the more common of these suggested approaches.

In the paper outlining the *agent environment web server extension* [65], it is stated that achieving host to mobile agent security (i.e. mobile agent protection) is ambitious and difficult generally. Therefore they designed their mobile agent system as a net of trusted hosts (only with support for inter-host authentication and mobile agent transport integrity) as they concluded there is no satisfying solution for the mobile agent malicious agent platform problem, thus defining away that problem. This approach might be useful in some circumstances, however our research focuses on exploring solutions which are more widely applicable. If the promise of large numbers of autonomous and safe mobile agents working in an Internet electronic services marketplace (to agent platforms whose trust level is often unknown to the mobile agent/mobile agent user) is ever to be realised, advancement of security solutions will be needed.

Cutdown trusted computing devices, like smartcards, with limited computational resources are geared towards protecting client-side applications, whereas the greatest threats arise from (agent platform) server attacks in the mobile agent paradigm. Moreover, as mobile agents can be of arbitrary code size and require arbitrary operating system services, approaches based on *tamper proof environments* [226, 235, 110] nestled in trusted computing hardware such as smartcards or *secure coprocessors* are not a viable solution for defence against the malicious
host platform in a mobile agent Internet infrastructure. In fact, it is not a safe assumption to make that just because secure coprocessors or tamper-resistant devices are employed they are any more secure than software only security approaches. Roth [166] describes a weakness in a secure coprocessor mobile agent security protocol approach which renders it weaker than similar software only protocols.

4.2.2 Mobile Agent Data and Code Integrity

Mobile agents by their execution nature are susceptible to malicious agent platform attacks against their data and code components. The aim should be to offer protection such that mobile agents can acquire new data on each agent platform they visit (moreover, their code should not be misused), with tampering of pre-existing data detectable by the agent user (and preferably for preserving their autonomous property) by other agent platforms in the agent’s itinerary. Most approaches for protecting the data and code components of mobile agents will not be able to prevent every attack but will at least provide some protection, often by the detection of discrepancies in an expected value/state based on a “reference” (non-attacking) value/state.

4.2.2.1 Execution Tracing

Vigna [212] presents an approach that allows a mobile agent owner (under certain assumptions\(^1\)) to detect any possible attempt to tamper with agent data, code, and execution flow. The proposed mechanism does not require dedicated tamper-proof hardware or trust between parties, both advantageous when designing a generic solution for mitigating part of the malicious host platform problem.

The approach of execution tracing is such that: each agent platform logs every action performed by a mobile agent while it is working there; and a cryptographic hash/fingerprint of the summary of the work performed in a mobile agent’s visit is submitted to an external entity (possibly the mobile agent home platform or a trusted third party) for post-mortem analysis of the data. Thus, the traces are used to perform program execution verification - specifically, checking the mobile

\(^1\)These execution tracing assumptions include: All agent platforms are participants in a public key infrastructure, a means for certifying interpreter integrity is available, and so too are methods for service billing and sanctioning of principals detected as acting maliciously or non-ethically.
agent against a supposed history of its execution. Such checking can detect acts of tampering, allowing the mobile agent user to prove that claimed operations performed by the mobile agent could never have happened, and ascertain where in a mobile agent’s itinerary (i.e. which visited agent platform) tampered with the mobile agent’s data and/or code.

This approach does initially appear to be a deterrent against malicious host platform behaviour, but there are a number of drawbacks with the scheme, including: agent platforms must compute and maintain large, non-repudiable log files; a secure protocol must be used for transferring the cryptographic hashes to external entities; time synchronisation across (potentially malicious) platforms is required; detection process requires manual checking, or software at the external entity to automate an intelligent analysis of the results from verified hash summaries; collaboration attacks between successive malicious agent platforms cannot be detected; and the approach as it is presented requires the agent platform to know the identity of the mobile agent (and possibly record other sensitive information about, or relating to, the mobile agent).

4.2.2.2 Reference States

To protect a mobile agent’s data integrity from attacks by their execution environments, the reference states protocol is another approach to detecting modification attacks. Reference states are mobile agent states that have been produced by non-attacking “reference” hosts [85, 86]. These reference states, thus, can be used at a reference host to detect discrepancies (arising from possible malicious behaviour by an agent platform that the agent has just come from) when a tampered agent’s state is compared to the prior reference state.

The reference states protocol appropriately modifies the execution tracing protocol approach for its own means. In contrast to the original tracing approach [212], this new protocol offers a model where the execution on one host is checked unconditionally and immediately on the next hosts, regardless of whether this host is trusted or untrusted (whereas in the execution tracing approach checking is post agent mission completion on a need-to-know basis initiated by the agent’s user). Given this real-time detection feature, the reference states protocol is advantageous over the execution tracing approach because it can trap malicious agent platform behaviour (specifically modification attacks) as soon as possible and limit the future damage a compromised agent can do to itself and
4.2. Countermeasures for Thwarting Threats to Mobile Agents

This modification preserves the qualitative advantages of mobile agents like asynchronous execution, but also introduces two new problems: (1) Input to the execution session on one host cannot be held secret to a second host, and (2) Collaboration attacks of two consecutive hosts are possible. Via a simple prototype, it was found (as would be expected) that generally, the overhead needed for the reference states protocol roughly doubles the cost of mobile agent execution without this security mechanism [86]. Some optimisation (by skipping reference state checks for trusted hosts) is possible, as detailed in [85].

We utilise and adapt Hohl’s reference states protocol in our MASHIn strategy. Concrete details on Hohl’s protocol and our extensions are the subject of Chapter 8 of this thesis. Among the benefits that come from our changes are the elimination of the two abovementioned problems with the reference states protocol.

4.2.2.3 Partial Result Encapsulation

Partial result encapsulation is outlined by Yee [237]. The idea of a Partial Result Authentication Code (PRAC) is very similar to that of a Message Authentication Code (MAC). Instead of authenticating the origins of a message, a partial result encapsulation demonstrates the authenticity of an intermediate mobile agent state or partial result that resulted from running on an agent platform. Similar to MACs, PRACs are cheaper to compute than digital signatures and have slightly different security properties.

The property that PRACs ensure, according to Yee, is forward integrity: if a mobile agent visits a sequence of agent platforms, none of the partial results generated at agent platforms up till the first malicious agent platform encountered can be forged. This is achieved by nested digital hashing of partial results from agent platforms. The forward integrity property is also present in the execution tracing mechanism [212] approach, and to a lesser extent\(^2\) in the reference states approach. Others claim that the PRAC approach only provides weak forward integrity and improvements to the partial results protocols introduced by Yee have been made [227, 109, 233].

Yee lists three methods for generating PRACs [237]: “Simple MAC-based PRACs”, “MAC-based PRACs with one-way functions”, and “Publicly verifiable

\(^2\)Due to the possibility of collaboration attacks in the reference states protocol approach.
PRACS”. The common feature of these three methods is that the integrity key associated with the current agent platform is erased by the mobile agent prior to its migration to the next agent platform in its itinerary (i.e. the most crucial security difference to MAC keys which are not subjected to multi-hop platform executions in securing client-server applications).

PRACs would require a trusted third party to time-stamp a digital fingerprint of the results for any legal assurance. Moreover, digital signatures are not mandated by Yee’s partial result encapsulation approach, which leaves open the possibility of repudiation by agent platform entities. A problem with the partial results scheme is that it requires the results to be checked by the agent dispatcher (usually the agent user), breaking the autonomous property of mobile agents and not limiting damage to agent users and possibly agent platforms (from a compromised agent).

4.2.3 Summary of Mobile Agent Data and Code Integrity

Execution tracing, reference states, and partial result encapsulation are leading approaches to assist in ensuring that the execution and performed computations of mobile agents are not maliciously manipulated by attacking agent platforms. However, if such an attack does occur it should be readily discernible, reported, and/or dealt with in an appropriate manner. The overheads and post mission non-autonomous checking in execution tracing (discussed in Section 4.2.2.1) makes it a largely undesirable mechanism of choice. Partial result encapsulation (discussed in Section 4.2.2.3) provides some nice purported features such as forward integrity - though one can see obvious problems with the robustness of this scheme, and the veracity of the purported forward integrity property has been questioned by others. The reference states mechanism (discussed in Section 4.2.2.2) seems most promising as a general concept because the checks are performed in real-time (i.e. during the agent’s itinerary immediately after an untrusted agent platform), but suffers from two significant shortcomings. We alleviate these two problems and extend the reference states checking mechanism in Chapter 8. Other approaches for protecting mobile agent data and code integrity exist, such as proof verification by holographic proofs [237, 19], but remain far too much in the theoretical spectrum to be considered viable alternatives in the near future. In the next sectional part we look at state-of-the-art approaches for protecting mobile agent data privacy.
4.2.4 Mobile Agent Data Privacy

The desired property of data privacy for mobile agents is very difficult to achieve in the generic sense, but can be mitigated by either (i) distributing sensitive data components to a number of mobile agents or distributed locations on the one hand, or (ii) making (partially) illegible the semantics of a mobile agent’s data and code for execution to an interpreter.

Considering the strategies employed in the following data privacy approaches are intriguing from a mobile agent security perspective, as the techniques are regularly the foundation of other strategic approaches in mobile agent protection. Furthermore, they help to highlight recurring problems or limitations which arise in many agent security approaches.

4.2.4.1 Cooperating Agents with Mutual Itinerary Recording

In the cooperating agents with mutual itinerary recording approach [164], the idea is to decrease the susceptibility of undetected malicious host attacks by distributing data and operations across mutually supporting mobile agents with carefully selected itineraries, thereby minimising the likelihood of them being located/executed at colluding malicious hosts simultaneously. Hosts, thus, must collude to ensure substantially damaging attacks (by reconstructing this fragmented data and code logic across disparate mobile agents).

The rationale behind this scheme, and other similar data/code splitting multi-agent schemes [134, 237, 147, 192, 126], is founded on the assumption that (at most) only a few agent platforms are malicious, and even if a mobile agent encounters one, the agent platform is not likely to collaborate with another malicious agent platform being visited by the peer. Therefore, by dividing up the operations of the application between two mobile agents, certain malicious behaviour of an agent platform can be detected [94, 164].

It may, however, in the general case be difficult to ascertain appropriate mutual itineraries - at least one of the mutual supporting mobile agents should be located at a trusted, non-attacking agent platform at any one time; and mutually cooperating agents should not both visit any one same agent platform throughout their itinerary lifetime. As the scheme relies on the assumption that no pair of hosts chosen from both mobile agent’s itineraries collude, this could be a significant limitation in an agent’s (possibly dynamic) work itinerary.
4.2.4.2 Environmental Key Generation

In the environmental key generation [162] approach, a mobile agent’s enciphered code is unlocked randomly, by reliance on an environmental condition trigger (such as a search string being found). The simplest “clueless” mobile agents look for their activation keys on a fixed data channel, for example usenet news groups, web pages, or email messages. The environmental condition can be hidden through a one-way hash. Reading the mobile agent’s code cannot reveal the triggering message.

Serious problems with this approach can be seen, though, including mobile agent execution environments (e.g. Java based systems - the most prevalent language used to program mobile agents) typically disallow the dynamic creation of code at run-time for execution (for host security reasons), and the agent platform could simply read the data code out upon the triggering of the environmental condition (by polling the mobile agent, or employing a state event trigger notification mechanism), instead of executing it [94].

4.2.4.3 Computing with Encrypted Functions

In computing with encrypted functions, the aim is to ensure mobile code privacy such that:

Alice has an algorithm to compute a function \( f \). Bob has an input \( x \) and is willing to compute \( f(x) \), but Alice does not want Bob to learn anything substantial about \( f \). Moreover, Bob should not need to interact with Alice during the computation of \( f(x) \) [174].

In more generic terms, the goal is to find a method which allows mobile agents to safely compute cryptographic primitives, such as a digital signature, even though the code is executed in untrusted computing environments and operates autonomously. The approach employed is to have the agent platform execute a program embodying an enciphered function without being able to discern the original function; the approach requires differentiation between a function and a program that implements the function [94]. The computing with encrypted

---

3Mobile agents secured using the environmental key generation approach are tagged “clueless” by the scheme’s authors for a few reasons: (1) From a blackbox perspective, their logic is immediately indecipherable, (2) From the agent’s perspective, it does not know its purpose until its code is deciphered, and (3) Such agents might reasonably be likened to the sleeper agents of the “Manchurian Candidate” and other Cold War era spy films.
functions proposal shares similar goals as that of much earlier work conducted on privacy homomorphisms [171], though that work has also proved very difficult to transparently incorporate in generic applications. The ability to transparently "plug and play" security measures is a very important criterion when one wishes to protect application-independent mobile agents, fundamental to our intentions.

A theoretical positive of this computing with encrypted functions approach is that, in theory, it is non-interactive - meaning it adheres to the generic goal for protecting agents - no interactive communication is needed between agents, at remote agent platforms in its itinerary, and its home platform.

Notwithstanding this, serious limitations hinder this proposal. Cryptographic theory has yet to find encryption schemes for computing arbitrary functions (without substantial knowledge attainment of the function) in a non-interactive manner\(^4\). No information about the encrypted computation must leak to the agent platform and only the originator may receive any output, thus eliminating active mobile code that performs some immediate action on the agent platform (like putting a hold on a shopping item) [233]. Moreover, the scheme does not prevent a number of other attacks such as replay, or modification in multi-hop mobile agent systems.

As Posegga and Karjoth point out [154]: An agent is not an individual, in the sense that it is separated from its executing environment. To the visited agent platform, all details and internals of an incoming agent are exposed. They describe the concept of the agent and the agent platform "merging" when the agent is executed: the agent becomes the agent platform and vice versa. As a consequence of this agents cannot carry any secrets, which is why in simplified terms, applying cryptography to protect agents while executing on agent platforms - including the computing with encrypted functions approach - fails to offer satisfactory practical solutions at this time.

4.2.4.4 Time Limited Blackbox Security

Another approach for achieving data and code privacy is to use code mess-up algorithms (more commonly termed code obfuscation [118]) to make unintelligible

\(^4\)It is claimed that attaining execution secrecy of arbitrary functions is possible by utilising threshold cryptography and distributed computers [16]. However, such a proposal if applied to generic mobile agents would complicate the issue of choosing trustworthy hosts, result in likely unacceptable agent mission execution times, and contradict the desired autonomous and dynamic itinerary properties of mobile agents.
the mobile agent platform’s immediate perception on the role and data of the mobile agent, and (in the time limited blackbox security approach) where possible restrict the useful lifetime of the mobile agent’s code and data [83, 84].

The mess-up algorithm is responsible for generating a “new” mobile agent out of the input original mobile agent. As a result, the original mobile agent’s code and data representation differs from that of the “messed-up” mobile agent - making it more difficult to reverse engineer, but maintaining the same behaviour. To prevent dictionary attacks, the mess-up algorithm must be given a random parameter that allows the algorithm to create different (random) agents out of a single original one input into the mess-up algorithm.

The type of a mobile agent statement can be hidden until the statement is executed by dynamically creating it at runtime, if self-modifiable code is supported by the runtime environment - though the Java virtual machine prevents this capability. The location of a mobile agent statement in a code segment can also be hidden similarly.

The significant advantages of such an approach is that it can be very effective for relatively short-term secrets, and no cryptographic keys or algorithms are required.

However, looking at the converse side of things: all messed-up code and variable names can eventually be reverse-engineered, regardless of the complexity of the obfuscation algorithms; no blackbox algorithms exist that work for arbitrary input data (only polynomials and rational functions) \(^5\); and there is no formal model for determining the relative strength of code mess-up algorithms (i.e. for quantifying the directly proportional useful protection period) [48, 94].

NAI Labs’ proposed concept of self-protecting mobile agents, described in Section 4.2.4.6, relied heavily on code obfuscation techniques for its purposes.

4.2.4.5 Secret Splitting Scheme

With little doubt, the most challenging task for mobile agents is to remotely compute with a secret key (i.e. “in public”). Digitally signing a document, for

\(^5\)Goldreich [73], and Goldreich and Ostrovsky [74], argue that it is theoretically possible to create blackbox algorithms that work for arbitrary input data - but, they rely on the assumption of an underlying tamper-proof CPU. From our perspective of a mobile agent platform this assumption is unacceptable. Indeed, Goldreich - as co-author of another later paper [14] - concludes it is theoretically impossible to create arbitrary blackbox algorithms using standard computers (i.e. with unshielded CPU/memory hardware components).
example signing off on an e-commerce contract is one such computation that may need to be carried out by mobile agents.

Onbilger, Chow, and Newman [147] propose a multi-agent model together with a simple *secret splitting scheme* for signing with shares of a key carried by members of a group of mobile agents cooperating to accomplish a single task without the necessity of reassembling the key. Their work builds on much earlier work on threshold cryptography [49, 50, 68], this time specifically for the purpose of computing a digital signature by co-operating mobile agent shares “secretly” over distributed computers.

In addition to the well-known multiplicative and additive properties of the RSA algorithm, similar techniques with the El Gamal public key cryptosystem are demonstrated to show their applicability for secret splitting and digital signing for mobile agents. The advantage of the techniques presented in this approach is that they use nothing but the original signing and verification algorithms of RSA and DSS.

The secret splitting scheme seems, to some degree, disingenuous because only one of the split mobile agent parts (and hence key part) needs to break for the signature generation (and hence transaction) to fail. If combined with robust threshold cryptography techniques [68, 63] then greater robustness could, theoretically, be added to the signature function completing successfully - specifically only a threshold $t$ out of $n$ replicated mobile agents carrying the pertinent key parts would be required. But, to consider an example, if the key needs to be split in five distinct parts (each part individually carried by a mobile agent) then one can easily see how creating $n*5$ mobile agents, for robustness purposes, will create significant additional execution time and network overhead to an already costly multi-agent scheme.

The risk of course could also be mitigated be having the split mobile agent parts only residing on trusted platforms. Moreover, as the authors claim, the secret splitting multi-agent model provides a high level of security if the distributed signature function succeeds as it would be difficult to compromise all of the remote mobile agents, which together form an autonomous group, especially if some are located at trusted, non-attacking agent platforms. However, it may be problematic to select an appropriate itinerary for the mobile agent parts, and there is likely a significant additional mission execution time and network transport cost (for both the travel of the mobile agent parts and for communicating...
4.2.4.6 Self-protecting Mobile Agents

NAI Labs’ Self-Protecting Mobile Agent (SPMA) research project [126] aimed at developing strong protection for mobile agents by empowering them with the capability to protect themselves (autonomously) from maliciously-acting agent platforms. Self-protecting agents could thus perform their missions with higher confidence even though they sometimes roam through unknown and possibly hostile territory.

This NAI Labs’ objective was to be investigated by combining three core techniques in formulating *self-protecting mobile agents* [126]:

- **Distributed Mobile Agent State**: Each mobile agent will be partitioned into a set of communicating programs (known as agentlets) executing on independent hosts. Critical information will be spread across the agentlets, thus limiting vulnerability to any proper subset of the hosts. This is similar to a merging of strategies from the mobile agent code and state component division in the keylets approach and mutual cooperating agents approach [192, 164] (discussed in Sections 4.2.4.7 and 4.2.4.1 respectively).

- **Obfuscation with Periodic Regeneration**: Executable code and data of each agentlet will be obfuscated using a variety of techniques (e.g. randomly selected, but equivalent, algorithms and data representations). Obfuscation will be used to ensure a host that chooses to execute a mobile agent cannot subtly-modify the mobile agent, for a period of time. Obfuscation will delay, not prevent, corruption or brainwashing of mobile agents using reverse engineering. It is intended to extend this limited protection by dynamically generating newly obfuscated versions of mobile agents, and removing older versions from service. The regeneration periods will be set so that a successful attack on an agentlet cannot be accomplished before the agentlet expires. This is similar to the time limited blackbox security approach [84] (discussed in Section 4.2.4.4).

- **Monitoring and Recovering**: Agentlets will be self-monitoring and will also monitor other agentlets. Using challenge/response techniques, mobile agents will automatically exclude compromised agentlets, report the identities of the signature protocol between the distributed shares) overhead.
tampering nodes, and replace lost agentlets. This sounds like it is an extended concept on the mutual cooperating agents approach [164] (discussed in Section 4.2.4.1).

Whilst the planned three-pronged self-protecting mobile agent approach initially seemed promising (and clever) in theory, the practicality of realising such a solution was hindered by significant technical challenges requiring non-trivial state-of-the-art advancements. The most problematic was deriving protection guarantee periods for non-trivial obfuscation (which was a central idea in the time limited blackbox security approach discussed in Section 4.2.4.4). The original goal of the SPMA project was to develop automated tools to protect mobile agents from attacks by malicious hosts. In development of those tools, NAI Labs realised obfuscation could not be relied upon to give a reasonable amount of security. Because of this, they re-directed their SPMA project focus solely to studying obfuscation [48].

4.2.4.7 Keylets

Keylets [192] are mobile code (not mobile agents per se) used to control the propagation of keys in a distributed system, as well as a technique for enabling mobile agent code secrecy that involves encryption of partitioned code components. Keylets are used to support this technique by directing the distribution of keys that decrypt the encrypted components.

The propagation of keys provides a possible mechanism to implement trust propagation as part of an overall trust model for a mobile agent system. The development of such a trust model could provide an efficient way for existing code security techniques to be deployed appropriately (with due consideration to performance) on the basis of trust relationships between different entities in a mobile agent system.

A fusion of the trust management capabilities from the keylets approach with the distributed nature of cooperating mutual agents (discussed in Section 4.2.4.1) could assist in the secure data repository, data transfer, and communication between mobile agent entities.

The keylet approach involves partitioning mobile agent code and state information into self-contained components. These components are then encrypted using symmetric keys which are subsequently made available to agent platforms that will host the mobile agent on a needs basis.
Chapter 4. Existing Strategies for Protecting Major Mobile Agent System

As with other approaches, the keylet approach is not sufficient to thwart a wide range of problems associated with the mobile agent malicious host platform problem. It is, rather, a preliminary attempt at a possible key management solution for securing the division of, and deployment of, important code segments and state information within a mobile agent to target platforms. In effect, each component represents a different security level of the overall code and state that needs to be safeguarded. The keylet idea is very much associated with the concept of trust management, where authorisations to perform specific (mobile agent code) actions are associated with encryption keys which are propagated throughout the system.

4.2.4.8 Supervisor-Worker Framework

The Supervisor-Worker Framework approach [61] presents an elegant and innovative higher-level solution for part of the malicious host problem aspect in mobile agent security. The framework limits the risks of leakage and tampering as the data stored in the supervisor will never be accessible to potential malicious hosts, since it will only reside on trusted hosts.

The Supervisor-Worker Framework includes the security measures as an integral part of the design of mobile agents. Thus, common problems of retrofitting security into an existing application do not arise.

The framework consists of a coordinating entity (the supervisor) and several independent entities (the workers). The workers move from host to host and try to finish the tasks they were assigned from the supervisor. The supervisor holds all the current knowledge found by the workers and uses this knowledge to accomplish its task. If it needs more information it creates a new worker and sends it to agent platforms to get this information.

The key difference to the client-server paradigm is that the supervisor component is a mobile agent as well. So it can move to a host near the area its workers will operate in. There is an important prerequisite: the supervisor must exclusively visit secure trusted places. In the worst case this is the host system where it has been initialised (likely the home agent platform of the mobile agent user). It is demonstrated that this framework solves special aspects of mobile agent security, including avoiding the possibility of information eavesdropping. Tampering well-designed worker mobile agents does not reveal any confidential information.
Besides the main intent to make mobile agent technology more secure, the Supervisor-Worker Framework provides additional benefits and boosts some of the mobile agent’s advantages due to its design and structure (e.g. flexibility and separation of concerns). Like any mobile agent framework it must obey the requirements of mobile agents. Otherwise, the inherent advantages of mobile agents - like autonomous and asynchronous execution, platform interoperability, etc. would have been lost. These requirements are met by the framework because of the underlying Master-Slave pattern. Its separation of code focusing on coordination and code focusing on computation make the pattern the ideal basis for the framework. The design (enabled due to the Master-Slave pattern) allows easy integration of this framework in applications and eases porting to other mobile agent systems.

As noted, in the worst case when there are no trusted sites available for the supervisor to coordinate things from, the supervisor must reside and work from its home (agent owner’s) platform - but, what if the home platform wishes to go into disconnected mode? The framework then falls apart. Regardless, it breaks our (and the general) mobile agent definition - autonomous work done solely at remote agent platforms.

Another drawback of this framework proposal is the significant time involved in preparation, for example the high-level non-trivial nature of the decision making involved in defining constraints for the supervisor mobile agent, and/or in real-time, to ensure the safety of his workers (and ultimately the mobile agent’s mission). This places a significant extra - quite probably - unwanted burden on the mobile agent’s designer and/or user.

4.2.5 Summary of Mobile Agent Data Privacy

Ensuring mobile agent data (including code) remains protected from prying “agent platform eyes” is very difficult in the general case. This is due to the fact that, as an inherent property of the mobile agent paradigm model, agent platforms have access to the agent’s code as it is interpreted. They may even have access to the code after the agent has finished its execution on the agent platform and migrated, if an agent platform acts maliciously and keeps copies of the agent.

Current state-of-the-art approaches offering mitigation to this threat often distribute sensitive data components to a number of mobile agents located at distributed sites. The cooperating agents with mutual itinerary recording (discussed
in Section 4.2.4.1), secret splitting scheme (discussed in Section 4.2.4.5), self-protecting mobile agents (discussed in Section 4.2.4.6), keylets (discussed in Section 4.2.4.7), and the supervisor-worker framework (discussed in Section 4.2.4.8) schemes employ this tactic. If appropriate multiple itineraries are selected including trusted agent platforms, such schemes can work well - though often choosing appropriate multiple itineraries is not a trivial task, and the availability of trusted agent platforms cannot be assumed. A drawback of these approaches is an obvious increase in network traffic, per the multiple distributed data components.

The other tactic employed in offering mitigation to this threat is to employ either cryptographic mechanisms (like in the environmental key generation and computing with encrypted functions approaches, discussed in Sections 4.2.4.2 and 4.2.4.3 respectively), or code-mess up algorithms (like in the time limited blackbox security and self-protecting mobile agent approaches, discussed in Sections 4.2.4.4 and 4.2.4.6 respectively). Pure cryptographic mechanisms are fundamentally not well-suited to protecting risks from the malicious host problem because they can be circumvented (such as in the environmental key generation approach), or are inflexible in practical applications because they remain largely theoretical (such as in the computing with encrypted functions approach). Obfuscation approaches, like the time-limited blackbox scheme, offer only short-term protection because their mess-up primitives are reversible.

4.3 Measures to Unite MA Users and MA Platform Owners

We have come across no approach that deals with mobile agent stakeholder relationships or which conceptualises a mobile agent system trust model that explicitly attempts to bring the two major mobile agent system stakeholders (i.e. mobile agents users and mobile agent platform owners) closer together and/or form a working relationship.

4.3.1 Social Control for Secure Internet Commerce

We came across an early use of social control, by Rasmusson and Jansson, in decreasing attacks involving agents [156]. This work looked into discouraging dishonest or irrational agents in open markets by having the participants them-
selves responsible for the security, as opposed to leaving the security to some external or global authority. Social mechanisms do not deny the existence of malicious participants, but instead aim at avoiding interaction with them. They argue that in contrast to “hard security” mechanisms (which aim to provide watertight protection) such as passwords and firewalls that are totally compromised if somebody finds a way to bypass them, “soft security” permits anything as long as it is good behaving. Their work raised some pertinent recurring problem questions in user-issued reputation systems:

- How are new agents/entities introduced into the system, when they do not have a reputation?
- Can an agent/entity build up a good reputation (either fairly or otherwise) and then abuse this to commit crimes (without punishment)? Conversely, what happens if an agent/entity is tarnished wrongly with a bad reputation and is hindered from doing business?
- Would the practice of formulating an entities reputation directly from peer (possibly unfairly targeted) statements, and thus placing the feasibility of their business success in jeopardy, be legitimate in terms of the law?

Rasmusson and Jansson go on to discuss three possible mechanisms for social control. These are inspired by establishing opinions about other group members which are either local to each individual, distributed, or trusted to a third party. In the latter case, they comment that reputation can be used in self-improving systems where the reputation corresponds to how well a service is performed. Later, in Chapter 9, we will show how this is the basis of the main-level reputation possibilities in the MASHIn. We feel this trusted-third party reporting of actors (agent platforms, for example, in the MASHIn context) whose services perform badly compared to others - thus encouraging use of services offered by actors with better reputation - is a process which is least vulnerable to improper use of reputations.

Rasmusson and Jansson’s work simulates agents from the perspective of the agents being participants in an e-commerce market environment, not specifically representing agent users and and agent platform owner stakeholders per se. In this respect, their work reflects the approach of others focusing solely on agent-to-agent trust management - such as [24], [148], and [102].
For practical advancements to be seen in the relationships between mobile agent users and agent platform owners (and thus also the future prospects of Internet mobile agents), one must conceptualise trust mechanisms which focus on, and accommodate a lowering of, concerns for both stakeholders. As noted in Section 3.2, we see this as the current great mobile agent security stumbling block which must be overcome for mobile agent stakeholder interaction to have a real chance of proliferation.

4.3.2 Trust Relationships in a Mobile Agent Infrastructure

The importance of formalising a trust model for the stipulation of initial mobile agent system stakeholder relationships, and the inference of new trust relationships in a mobile agent infrastructure were highlighted by Tan and Moreau [193].

Tan and Moreau’s mobile agent system code security trust model is an extension of traditional distributed authentication methods employed in a public key infrastructure. This model conceptually employs Vigna’s execution tracing code security mechanism [212] (discussed in Section 4.2.2.1) to stipulate initial basic binary relationships involving agent owner platforms (“agent users”), agent platforms, and verification servers. The verification servers are the entities in Vigna’s execution tracing mechanism that would, post mission triggered on suspicion by the agent user, verify a mobile agent’s execution traces. The execution traces are cryptographically protected and supplied by the agent platforms.

Derivation of new trust relationships are provided based on inference principles. The goal is to, from the initial set of trust relationships, generate a set of tuples that describe agent owner-agent platform trust relationships that exist between any given agent owner platform and all other agent platforms in the system.

One limitation of Tan and Moreau’s model is the assumption that all mobile agents in the system can be composed from a combination of a set of predefined code and state components made available by third party code producers. This prevents the capability of employing in-house built components and dispatching end-user programmed agents.

The key to evolution of the trust relationships in Tan and Moreau’s framework is the verification of an execution trace submitted by an agent platform to a verification server. The moment a verification server detects an invalid trace, the
4.3. Measures to Unite MA Users and MA Platform Owners

nature of its current trust relationships with the offending agent platform will be altered. This could range over several possible alternatives: severing all existing trust relationships, severing some trust relationships or degrading existing relationships by reducing the number of components that the relationship(s) is valid for [193].

Tan and Moreau’s approach does nothing directly to bring mobile agent users and mobile agent platform owners closer together. It indirectly may form new agent user-agent platform relationships but this trust relationship is only unidirectional. There is no practical use in deriving such relationships if the agent platform does not trust the agent user. As pointed out in Section 4.2.2.1 the execution tracing approach has many significant limitations. Building on this mechanism and adding dynamic trust relationship updates (involving multiple non-local entities) does seem to us to be a costly proposition, especially given only the formation of new unidirectional relationships.

In Chapter 9 we discuss in depth our conceptual mechanism for drawing the two major mobile agent system stakeholders closer together and to form new working relationships. One feature of our approach is that personalised stakeholder statistics gathered by TTPs and used to derive new relationships are more localised (i.e. involve less parties) and may be periodically updated - which can allow an easing of update costs.

4.3.3 Denial-of-Service Investigation and Trust Building

Cubaleska, Qiu, and Schneider [45], in designing a protocol to detect denial-of-service attacks from visited malicious agent platforms, have also permitted a procedure to allow the mobile agent user to define an appropriate order in which selected hosts should be visited, or the instruction of which hosts the mobile agent should not contact again. They have, thus, formed the basis of a trust mechanism based on positive and negative agent platform visit experiences.

In order to convict a guilty malicious agent platform (who has killed an agent), the proposal allows the agent user to become a detective, i.e. it enables the agent user to identify the host who performed an attack by the pre-application of well-known cryptographic primitives and a set of rules. Furthermore, the proposal ensures that an agent platform cannot be excluded from the agents journey.

The proposed Denial-of-Service Investigation and Trust Building protocol comprises two stages - a sender and a receiver protocol and an investigation
procedure. The first stage allows a procedure of sending agents and returning confirmation between a sender and a receiver. Any malicious denial-of-service behaviour by an agent platform will be identified by using its confirmation stored by its predecessor in the second stage, the investigation procedure [233].

The fact that the investigation is coordinated at the home platform, triggered on the real-time suspicion of the mobile agent user, contradicts our goal of safe autonomous execution for mobile agents completed in their entirety remotely - relieving the mobile agent user of mission interaction and detective responsibilities. Moreover, once more, the initial state trust and update mechanism is only unidirectional, from agent user to agent platform owner, not also from agent platform owner to agent user.

4.3.4 Summary of Measures to Unite the Major Stakeholders

There are no known schemes in the mobile agent security research literature which explicitly address the mutual mistrust between the major stakeholders (i.e. mobile agent users and mobile agent platform owners). This complexity is an inherent limitation to the practical prospects of the Internet mobile agent paradigm.

The social control for secure internet commerce scheme (discussed in Section 4.3.1) is a good example of a lot of investigative work on trust relationships concerning agents, in the respect that it only touches on inter-agent trust relationships. The fact that it uses soft-security mechanisms (specifically peer-reflected reputation statements) to calculate trust decisions for agents is interesting. We use reputations as part of our secure Internet mobile agent conceptual strategy, discussed in some detail in Chapter 9. However, we take a unique and more holistic approach whereby we consider the interests and reputations of both mobile agent system stakeholders in bringing these two parties closer together.

The trust relationships in a mobile agent infrastructure proposal (discussed in Section 4.3.2) is a mechanism for gaining trust in agent platforms from a track history of safe interpretation of agents. It suggests building on the execution tracing mechanism provided by Vigna (discussed in Section 4.2.2.1). Once again, this proposal is only a unidirectional major stakeholder trust mechanism - it does not address uniting the stakeholders, by overlooking consideration of the concerns of agent platform owners from risks faced by malicious incoming agents.

The denial-of-service investigation and trust building scheme (discussed in
Section 4.3.3) is another scheme which aims to build confidence for agent users that agent platforms, this time, will not deny service to their agents. Along with the trust relationships in a mobile agent infrastructure proposal (Section 4.3.2), the underlying mechanism for formulating trust data is triggered on the agent user’s suspicion of an attack. This somewhat undermines the intended autonomous nature of agents, freeing the agent user of responsibility for security checks and allowing them to be disconnected from the network.

In our secure Internet mobile agent strategy, we save the agent user the onus of ensuring their agent’s mission is performed safely, or if there is an attack they are not responsibility for detecting it. Moreover, our trust data gathering is performed regardless of whether their is suspicion and assists both major stakeholders - not just one or the other - to find respectable entities to interact with (i.e. mobile agent users - more specifically their agents - for agent platform owners, and vice versa).

4.4 Chapter Summary

In the previous chapter we highlighted a number of significant security issues tied to the Internet mobile agent system model. We identified and provided an overview of a large number of security threats targeted at the two major mobile agent system model stakeholders (i.e. mobile agent users and agent platform owners) that will be at least partially addressed in our high-level strategy proposed in this thesis for bettering Internet mobile agent prospects.

In this chapter we looked at a reflective sample of the commonly-referred state-of-the-art approaches to countering the threats in mobile agent systems. In Section 4.1 we provided an overview of the state-of-the-art countermeasure approaches for protecting mobile agent platform owners and their agent platforms. In Section 4.2 we provided an overview of the state-of-the-art countermeasure approaches for protecting mobile agent users and their mobile agents.

Tables 4.1 and 4.2 are our assessment of the countered threats detailed in the previous chapter. Please note that the second table is a continuation of the first table, but provided as two tables for easier formatting. The eighteen existing countermeasure approaches studied are common to both tables. The difference in the two tables is the threats addressed. As can be seen from our assessment of threat mitigation (partial or full) coverage in Tables 4.1 and 4.2, the vast ma-
The majority of the state-of-the-art approaches for mobile agent security only focus on mitigation of concerns for one stakeholder, either mobile agent platform owner (APO) or agent user (AU). The large white spaces (i.e. empty table cells) in these tables are reflective of the piecemeal focus coverage of the existing countermeasures\(^6\). The approaches usually only address one or two threats for a particular stakeholder. None of the reviewed approaches addresses a significant number of threat concerns for both stakeholders. By being piecemeal, and especially by not considering the security concerns of both major stakeholders, the existing countermeasures tend not to be easily interoperable, and there is no clear direction towards a uniform framework for deploying safe autonomous mobile agents.

As noted in Section 3.2 and discussed in some depth in Section 4.3, a great void in the state-of-the-art mobile agent security research work seems to be mechanisms to accommodate the concerns of both mobile agent users and agent platform owners, lowering their mutual inhibitions, and forming an environment conducive to interaction. Without such mechanisms, the promise of Internet mobile agents will remain largely a fantasy since both parties rely on the other for the mobile agent model to work.

Our work aims to address concerns for both stakeholders, importantly at the same time bringing them together in an environment conducive to interaction such that working relationships can be more easily formed. Integral to our high-level MASHIn approach is the employment of trusted-third parties (TTPs) capable of partially providing not only a large number of security services for agent users and their deployed Internet mobile agents, but also partially accommodating the concerns of agent platform owners for their agent platforms.

In the next chapter we start to look at how TTPs can be strategically utilised to mitigate threats in the mobile agent security problem space. We review the work of others in this endeavour, and precursor our strategic thinking on the employment of TTPs within the MASHIn - presented in-depth in subsequent chapters.

---

\(^6\) Mitigating inter-agent platform attacks targeted at the agent platform owner can be easily thwarted by using point-to-point cryptographic mechanisms such as SSL or IPSec, and this may explain why no countermeasure approach explicitly provided support for this.
### 4.4. Chapter Summary

| Denial-of-Service from Agents, APO, Difficult | P | P | P | P | P |
| Mobile Agent Gains Unauthorised Access, APO, Easy | P | P | P | P | P |
| Agent Refusal to Pay for Services Delivered to It, APO, Moderate | P | P | P | P | P |
| Infection via Malicious Agent Software Forms, APO, Moderate | P | P | P | P | P |
| Attacks Against Inter-agent or Platform Communications, APO, Easy | P | P | P | P | P |
| Impersonation from an Interacting Agent or Agent Platform, AU, Moderate | P | P | P | P | P |
| Agent Platform Denies Proper Service to Agent, AU, Difficult | P | P | P | P | P |
| Agent Platform Obtains Private Data or Mines the Agent, AU, Difficult | P | P | P | P | P | P | P |
| Eavesdropping of Agent Messages, AU, Difficult | P | P | P | P | P | P | P | P | P |

**Table Key:** P = Partially Addressed

**Table 4.1:** Summary of countermeasures coverage to threats (1 of 2).
### Chapter 4. Existing Strategies for Protecting Major Mobile Agent System Stakeholders

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AU, Difficult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent Platform Modifies the Agents Code or State</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU, Difficult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent Platform Manipulates Agents Control Flow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU, Difficult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent Platform Misroutes the Agent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU, Moderate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replay Execution Attack on an Agent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU, Difficult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent Platform Repudiation of Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU, Moderate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent Platform Repudiation of Execution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU, Moderate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent Platform Repudiation of Agent Migration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU, Easy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent Platform Returns Wrong Results</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU, Difficult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table Key:** F = Fully Addressed, P = Partially Addressed

**Table 4.2:** Summary of countermeasures coverage to threats (2 of 2).
Chapter 5

Utilising Trusted Third Parties for Internet Mobile Agent Security

It is not true that equality is a law of nature. Nature has no equality. Its sovereign law is subordination and dependence.

Marquis de Vauvenargues

In the previous chapter we looked at some of the major state-of-the-art countermeasure approaches which offer mitigation to the threats faced by the two major mobile agent system stakeholders (i.e. mobile agent platform owners and their agent platforms, and mobile agent users and their mobile agents). We observed that the vast majority of approaches aim to protect one or the other of the mobile agent system major stakeholders, and even then they usually only partially address one or two of the threats faced by that stakeholder. Furthermore, we highlighted that no known model or approach dealing with mobile agent stakeholder trust relationships investigates and eases the serious stumbling block for Internet mobile agent security that we have identified - i.e. mutually suspicious major stakeholders are precariously fearful and uncertain about interaction with the other because they are concerned that their next step may be a losing one.
In this chapter we take a look at how incorporating Trusted Third Parties (TTPs) can ease some of the security issues associated with Internet mobile agents. In Section 5.1 we review the work of others who have proposed incorporating TTP entities to reduce security concerns in the mobile agent research domain. We introduce a number of infrastructure-oriented TTP security services which can assist in ensuring safer passage for Internet mobile agents in Section 5.2, detail the advantages to each stakeholder when such a TTP security service is employed, and list some known schemes which incorporate similar TTP security services. In Section 5.3 we introduce the TTP component entities that are utilised in our strategy for securely enabling Internet mobile agents. Our strategy focuses on reducing the concern levels of both stakeholders, bringing them into an environment more conducive to business interaction. The chapter is summarised in Section 5.4, and we preview the presentation of our security strategy for mobile agents that constitutes the thesis remainder.

This chapter includes and builds on material found in peer-reviewed accepted papers [i] and [ii] listed on page xxi.

5.1 Existing Approaches

In the following subsections we provide an overview of how others have employed TTPs to assist in securing parts of mobile agent applications or standalone systems. The TTPs in these schemes are generally not well-suited to work with arbitrary other TTPs to provide viable solutions to the many unresolved security issues of concern to both stakeholders (discussed in Section 5.2) when considering support for possibly millions of agents and thousands of agent platforms within a multi-domain, multi-agent system, and application-independent mobile agent infrastructure.

Where deemed useful to our secure infrastructure-oriented strategy for enabling Internet mobile agents, we later adapt and incorporate some of this foundation mobile agent TTP work. We see our approach as an overall more comprehensive infrastructural approach and strategic vision for bringing the two major stakeholders closer together so that the theory of Internet mobile agents may transfer to a form where the practical-reality prospects of Internet mobile agents are bettered.
5.1. Existing Approaches

5.1.1 Privilege Management Scheme with Attribute Authorities

Jansen presents, over two papers, his ideas for a privilege management scheme for mobile agent systems by utilising attribute certificates for expressing access control rights policies [97, 95]. Discussion in one paper [97] focuses on which entities in mobile agent systems express security policy, what kind of security policy is expressed, how security policy is expressed, how security policy is conveyed, the rationale for attribute certificates, and a scenario is provided for using attribute certificate processing within mobile agent systems. In the other paper [95] the privilege management scheme using policy and attribute certificates is elaborated on, by providing an overview of the scheme - in particular policy bearing certificates, attribute certificate elements, policy processing, suitability and use, certificate representation, chained authorisations, criteria for applying the scheme to a mobile agent system, Java-based mobile agent systems and policy certificates, attribute certificates, and the policy engine.

Jansen and Karygiannis [97] describe the basic functional concept of the mobile agent privilege management scheme this way: As an agent moves among agent platforms, it carries along its issued attribute certificate(s) and the identity certificate(s) of the issuer(s) who assigned the privileges to this agent (e.g. user or other policy-setting authority like an Attribute Authority). A platform receiving the agent determines the relevance of the certificate(s), verifies the issuers identity, perhaps with the assistance of a Public Key Infrastructure (PKI), and determines whether compliance exists between the privilege assignment conveyed in the certificates and the platform’s prevailing policy. Policy-setting principals can assign privileges via attribute certificates by either jointly signing and issuing a single attribute certificate to an agent, or individually issuing attribute certificates to an agent and having them interpreted and validated by the policy engine of the agent platform.

The Attribute Authority (AA) TTP conceptual component is explicitly defined in the ITU X.509 public key and attribute certificate frameworks specification [90] as “an authority which assigns privileges by issuing attribute certificates”, with an analogous role to that of a certificate issuing Certification Authority (CA) in a PKI. Attribute certificates are employed by Chadwick et al. in the PERMIS Privilege Management Infrastructure (PMI) project [31, 57]. The EC funded PERMIS project was given the challenge of building an X.509 role based PMI
that can be used by very different distributed applications in three cities of Europe. The cities had prior experience in running pilot-PKI applications, and wanted to add a PMI capability. It is claimed that the PERMIS PMI is being utilised in four very different applications across Europe [31].

Our strategy for having large numbers of mobile agents capable of working with large numbers of agent platforms - with both stakeholders more confident in secure interaction - incorporates and builds on Jansen’s [95] proposed scheme incorporating attribute authorities for privilege management in mobile agent systems. Since the TTP attribute authorities are employed, the scheme does tend to scale well since it lessens the burden on local agent system administrators to deal with each agent’s specific privileges to agent platform resources. Furthermore, it allows policy to be expressed in a more generic manner which if, like in our privilege management strategy (to be discussed in Section 6.3), means that principal-to-agent-platform mappings can be naturally expressed and better managed - especially if the abstractions of role-based access control are employed.

### 5.1.2 SOMA Security Architecture TTP Entities

The Secure and Open Mobile Agent (SOMA) system was built as part of a research project to promote security and interoperability for mobile agents [43]. SOMA, a Java-based mobile agent system, provides a relatively rich and comprehensive solution for security problems associated with mobile agent system entities, including the two major stakeholders - agent users and agent platform owners [233].

Wherever possible, the SOMA security model has been implemented by taking into account the standard security solutions employed in distributed systems. This was seen as a key design principle because not only could the exploitation of ad-hoc security mechanisms require too great an effort, but more importantly non-standard tools are considerably less likely to be accepted in open environments. The SOMA framework supports the definition and the enforcement of flexible security policies to govern the interactions of agents with both other agents and with the available resources on agent platforms [205].

The definition of different locality abstractions allows the enforcement of layered security policies in which actions are controlled at both place (i.e. agent platform) and domain (i.e. collection of agent platforms) level. The domain defines a global security policy that imposes general authorisations and prohibitions;
5.1. Existing Approaches

each place can only apply restrictions to the domain-level set of permissions. We see the following TTP service responsibilities incorporated within the SOMA architecture [43]:

- **Policy Server**: Responsible for managing domain-level policy, specifically the activating and deactivating of policies having domain scope.

- **Domain Server**: Responsible for maintaining consistent references to the resources visible in the domain.

- **Roles Server**: Responsible for handling role management, including assigning new roles and updating existing roles for principals.

- **CA**: The Certification Authority is responsible for the issuing and life-cycle management of certificates.

Figure 5.1 provides a pictorial representation of the incorporated TTPs in the SOMA system architecture. SOMA supports both push and update of domain policies. In any SOMA domain one Policy Server is in charge of storing the domain policies. When a new place enters the domain, it may acquire the domain policy by sending mobile *Policy Agents* (PA) to the Policy Server for policy retrieval and successive installation. One approach to policy update is to rely on a stationary *Policy Monitoring Agent* PMA running on the Policy Server targeted at monitoring policy status; if policy changes a mobile *Policy Update Agent* (PUA) is created by the PMA to update places on the changing policies. The PUAs interact with the Domain Server to discover the current locations of places. If the update task of a PUA fails for some reason, a PUA can behave differently to recover from the failure: it can either notify the PMA of the error or figure out the origin of the problem or wait until the failure is recovered and then immediately propagate the policy changes. The Policy Server also coordinates with the Role Server when policies must be added or removed from the role specification [43].

With regard to credential management, agent and place owners join a SOMA domain through a two stage process - first an off-line process to be assigned identity and role credentials (via X.509 public key certificates, and via role certificates), and then an on-line process to validate certified credentials. At the place level, agents are permitted to access resources on the basis of the result from the intersection of domain and place policies [43].
Figure 5.1: The SOMA mobile agent system privilege management model centralised by domain (adapted from [43]).
5.1. Existing Approaches

We employ traditional distributed system TTPs like CAs\(^1\) (as discussed in Section 6.2) in our secure strategy for enabling a safer environment for mobile agent system stakeholder interaction, however we feel that SOMA’s central policy and role servers would not scale as well with our non-agent-system specific infrastructure-oriented approach capable of supporting potentially millions of agents and thousands of agent systems. The concept of roles (i.e. as used in role-based access control) is common to both the SOMA agent system and our infrastructure-oriented (see Section 6.3) approaches.

5.1.3 Tamper-resistant Trusted Processing Environment

Wilhelm et al. [225] propose the use of tamper-resistant hardware produced by a TTP hardware manufacturer. The Trusted Processing Environment (TPE) forms the complete execution environment of an agent platform which - if manufactured correctly - cannot be inspected or tampered with, and only executes a limited set of tasks administered by the tamper-resistant controlling module. The outside world cannot interact with the task of the tamper-resistant hardware except through a restricted interface which is under complete control of the tamper-resistant module.

The TPE proposal states that trust is an important issue in the context of mobile agents. Proponents of tamper-resistant device approaches take a pessimistic approach to trust that tries to prevent malicious agent platform behaviour rather than detecting and correcting such behaviour. The TPE approach relies on tamper-resistant hardware to provide mobile agents with the means to protect themselves [224, 225].

Drawbacks of tamper-resistant environments such as the TPE scheme proposed by Wilhelm et al. include their clash with the principles of open design and auditing scrutiny. Such approaches also limit scalability of applications - this is because the availability of such agent platforms (via agent platform owners willing to purchase and adopt TPEs) is limited at best, and TPE functionality is by its own intrinsic definition restricted to a cut-down set of instructions and filtered interactions [42]. As mentioned in Section 4.2.1, Roth discovered cryptographic protocol flaws in a hardware secure coprocessor proposal which can render the approach weaker than similar approaches that do not incorporate tamper-resistant

\(^1\)Of course, these also have their shortcomings; a list of these shortcomings is included in Section 6.2.
environments. This highlights the important point that approaches based on secure coprocessors or tamper-resistant environments are no guarantee for improved security [166].

In addition to the TPE approach drawbacks just mentioned, we do not prescribe that hardware-based secure solutions must be employed by any agent platform in our secure infrastructure-oriented approach (presented in detail from Chapter 6) because if one or the other of the major stakeholders did not trust the hardware manufacturer, an interaction would not be possible between these stakeholders. Hardware-based secure solutions may be useful in a closed mobile agent community compromising a limited-range of mobile agent applications, but the strategy we propose must be application-independent and draw together a larger community base of potential stakeholders.

5.1.4 Secure Mobile Agent Based Merchant Brokering

Karjoth [110] describes a distributed marketplace, where all servers are equipped with trusted but resource-constrained devices - specifically trusted devices like smart cards - to assist in the completion of a number of marketplace protocols. Three protocols for securely achieving the following shopping mobile agent tasks are discussed: (1) Use the trusted device only for executing cryptographic operations to seal and protect the results collected by the agent; (2) Employ devices that compare a given offer with the current best offer carried by the agent; and (3) Allow agents to download their own routines and data onto the devices - thus, agents get the capability not only to compare and gather information on prices of goods, but also to pick a satisfying offer and decide to buy it.

To improve the marketplace’s reputation, it may run under the supervision of a trusted organisation, for example a financial institution, a non-profit corporation, or a guild formed by merchants. Karjoth calls this organisation the market authority and assumes that it publishes guidelines of behaviour that are binding for its members. In case of a dispute between a customer and a marketplace member, it may also serve as an arbitration board. Karjoth stipulates that the market authority is a trusted third party for the customer as well as for merchants. Each merchants server (i.e. agent platform) is equipped with a trusted device (i.e. smartcard) that is provided by the market authority. The trusted device does not form the complete resources available at an agent platform but a small well-defined subset of common sensitive transactions.
We believe that the TTP concept of a market authority has merit for dispute resolution, but the detailed list of cryptographic operations and security risks that are associated with shopping mobile agents per se make them unfavourable with respect to cryptographic processing costs (both in terms of the number of primitive operations and key management issues). Furthermore, shopping mobile agents generally are more problematic than client-server querying and negotiation via a TTP broker, for example the former model introduces new risks such as denial-of-service from an agent platform which could result in loss of all previous offers and leave a mobile agent user application in a state of deadlock.

One solution by Kotzanikolaou et al. [114] proposes a system whereby there is one master agent and multiple slave agents. The master agent does not travel. The slave agents are mobile but they each travel only to one particular agent platform to negotiate, and they cannot complete a transaction. They return to the master agent with purchase contracts signed by the agent platforms. The master agent evaluates the signed contracts and presents the results to the user. Such approaches in essence avoid the malicious host platform problem in terms of shopping mobile agents, and the mobile agent user application is not left in a state of deadlock if one or more of the agent platforms denies service to individual slave agents [38].

If mobile agents are to be used for shopping purposes we believe such an approach is much more desirable (both in terms of efficiency and security) than the single roaming mobile agent which is dependent on a lot of different cryptographic operation (and questionable key management) measures and the robustness of agent mission completion is put in grave jeopardy.

Jansen and Karygiannis [96] back up our basic security concern premise attached to “window shopping” mobile agents by saying these mobile agents could visit vendor sites searching for the price and availability of goods and services, and when the mobile agent finds the goods and services that meet its criteria, a static agent at the home platform or at a trusted agent platform could complete the sale and sign the receipt with its private key. The loss of the window

2We are not yet convinced that shopping agents are a good example or sound potential application of mobile agents, though it seems the one which is (by far) most frequently detailed in the mobile agent security research literature. This latter fact could perhaps be explained due to (i.) without a ubiquitous framework for safely deploying agents, interest in devising more appropriate mobile agent applications will not be given due consideration, and (ii.) researchers want to keep their examples relatively simple, and consistent across the literature.

3But this would break the autonomous property of mobile agents.
shopping agent which has no authority to transfer money or sign receipts could be tolerated, but the loss or compromise of an agent that is authorised to sign documents or complete financial transactions is intolerable. Any evidence of the secure transaction, thus, remains solely on a TTP (agent platform) and is not generated or stored on the vendor’s agent platform.

The argument that mobile agent applications reduce network utilisation costs (and for lightweight devices this is an especially pressing requirement) may have been pertinent when mobile agents were first conceptualised in the early 1990s. Such a case is rare now, however, as the network bandwidth available to the average end-user today is many times larger and less expensive (so too are typical telecommunications costs). If a TTP broker is employed - for example in a mobile agent shopping application scenario - the costs are a small fraction of, and included in, the fee paid to the broker for using their services.

Whilst our secure strategy for bringing the major mobile agent system stakeholders closer together within an environment more conducive to interaction is application-independent, we employ TTPs in an important role of facilitating the safe execution of sensitive mobile agent operations and allowing critical mobile agent decision making processes to be completed in a non-conflicting environment. Our strategy is discussed in detail from Chapter 6, and a prototype application scenario (a purchase shopping mobile agent) is detailed in Chapter 7.

5.1.5 Stronger Shopping Agent Data Integrity Security Protocol

Yao et al. [234, 233] have employed a TTP for assisting in the design of stronger protocols for the provision of offer integrity for shopping mobile agents. The protocols of Yao et al. improve on the foundation set of security protocols for shopping mobile agents designed by Karjoth et al. in [109]. It is claimed that the “recoverable key commitment” technique overcomes a truncation attack involving colluding agent platforms in the shopping mobile agent’s itinerary who are able to remove offers from intermediate agent platforms and a stemming attack, which is closely related to the truncation attack, whereby one or more faked offers are inserted in the place of the truncated offer data.

The participants in the e-market setting proposed by Yao et al. include a buyer (i.e. the mobile agent user), a number of vendor servers (i.e. the mobile agent platforms), and a TTP that can be designated by authorities. The shopping
mobile agent enters the e-market through the TTP which may provide yellow-page directory querying services. The agent then travels from server to server and collects offers. After completing its journey, the agent returns to the TTP to verify its collected results and finally travels back to its originator. The TTP registers the vendor servers and manages the commitments from the registered servers. In case of a dispute between an e-market member and a customer or another member, the TTP serves as an arbitration board. The behaviour of the TTP can be publicly verified by a prover [234].

As explained in Section 5.1.4 “pure” shopping mobile agents are very problematic from a security viewpoint. Yao et al.’s work [234] on improving data integrity for shopping mobile agents addresses only two types (truncation and stemming attack mitigation) of the many security risks introduced when dealing with these shopping mobile agents (on top of the general security threats involving mobile agents). We re-iterate that using mobile agents for such applications is not an efficient or practical use of the mobile agent paradigm and should be avoided by using alternative models such as those discussed in Section 5.1.4.

We also present a purchase shopping mobile agent scenario implemented within the bounds of our conceptual strategy framework (see Chapter 7). Our strategy not only avoids many of the security problems with shopping agents per se (that were only touched on in this section) but is advantageous because our strategy is application-independent, ensures a more robust chance of mission completion for mobile agents, and promotes closer relationships and greater interaction between the two major stakeholders in the mobile agent system model.

5.1.6 Summary Review of Existing Approaches

In this section we have reviewed the use of TTPs in addressing various aspects of the mobile agent security problem-space. We discussed the following existing or prescribed roles for TTPs in the mobile agent security scene: attribute authorities, certification authorities, policy servers, domain servers, roles servers, trusted hardware manufacturers producing TPEs (i.e. agent platforms), market authorities distributing trusted devices like smartcards to agent platforms, TTP brokers for “hybrid” master-slave shopping mobile agents, and TTPs managing the commitments of “pure” shopping mobile agents.

In the next section (i.e. Section 5.2) we list generic TTP security services which can be helpful in relieving Internet mobile agent security problems. We dis-
Chapter 5. Utilising Trusted Third Parties for Internet Mobile Agent Security

cuss the benefits to both mobile agent stakeholders and provide a non-exhaustive reference listing of any known existing services or similar schemes. Then in Section 5.3 we take a preliminary overview look at TTP security entities present in our secure strategy for enabling Internet mobile agents, and briefly discuss their responsibilities.

5.2 Utilising TTPs to Solve Disparate Problems Between the Major Stakeholders

We cannot see the mass deployment of mobile agents in a global electronic services marketplace nearing reality without the incorporation of dedicated trusted third party infrastructural entities.

In the following subsections, we discuss a number of lingering issues with the practicality of safe mass deployment of mobile agents, in terms of the provision of security services. The important distinction from prior specific threat and mitigation discussion (respectively detailed in Chapters 3 and 4) is that the problems discussed in this section are general problems with direct implications to both stakeholders - whereas the prior threats and countermeasures were specific to only one stakeholder. If problems can be better addressed from the viewpoint of both stakeholders, they will be solutions more generic and readily usable because they will be inherently interoperable and provide an environment where each major stakeholder is more understanding and accommodating of the others’ concerns.

For each mobile agent infrastructural security issue, a TTP service for addressing the issue is given (where applicable, following in brackets is a cross-reference to further abstraction or detailed work relating to the proposed TTP service in our infrastructural solution), as well as the benefits to mobile agent users and agent platform owners in utilising the TTP service. We conclude each subsection with a brief, non-exhaustive reference overview of some existing proposals that incorporate such a TTP security service. Subsection 5.2.13 summarises the collective coverage of these generic TTP security services.
5.2. Utilising TTPs to Solve Disparate Problems Between the Major Stakeholders

5.2.1 Authentication

Problem Statement: How can agents and agent platforms be sure of whom they are interacting with (especially as the number of entities in the system grows considerably)?

Proposed TTP Security Service: Authentication Service with proxying TTP (see Section 6.4).

Benefits to Mobile Agent Users: Transporting of agents to spoofed agent platforms can be significantly mitigated.

Benefits to Mobile Agent Platform Owners: Only authenticated mobile agents are allowed onto their agent platforms. There can be a huge reduction in an agent platform’s key management concerns if it is mandated that agents wishing to work on agent platforms must pass through and be authenticated by the authentication server. This would result in the agent platform needing to know only the verifying (public) key/s of the authentication server TTP/s.

Known Existing Similar Schemes: Even though this would seem a relatively trivial mobile agent TTP security service to design, there are no known mobile agent systems or architectures which incorporate such a proposed TTP service accommodating both stakeholders. There are many agent systems or architectures which propose or utilise registration and/or certification authority functions but do not, as far as we know, unite those responsibilities with a genuine proxy authentication service to solve this problem (which in practical management terms grows as the number of entities - i.e. agent users and agent platforms - wish to interact within an infrastructure).

5.2.2 Mobile Agent User and Itinerary Anonymity

Problem Statement: How can an agent maintain the anonymity of its user and its travels?

Proposed TTP Security Service: Anonymity - User and Traffic - Service (see Section 6.4).

Benefits to Mobile Agent Users: Personal user identity and agent origin details are not disclosed to agent platforms. An agent’s list of visited platforms can also be kept secret from agent platforms (traffic anonymity).

Benefits to Mobile Agent Platform Owners: Since a mobile agent platform knows the agent is authenticated and audited by the TTP, there is no need for
it to see the agent user's personal details. It, thus, allows the agent to operate under the principle of least information.

**Known Existing Similar Schemes:** Yang and Nguyen's *Trust Third Party Checkpoint* (TTPC) structure proposal offers itinerary anonymity [232], and could be extended to offer user anonymity. SeMoA’s Atlas service for location tracking of agents provides some support for agent user anonymity [170].

### 5.2.3 Autonomous Mobile Agents

**Problem Statement:** How can agents completely perform their work autonomously (and securely)?

**Proposed TTP Security Service:** Disinvolving Mobile Agent User - Autonomous - Service (MASHIn SCC4MAP, see Section 7.3.2, and RECDAM, see Section 8.3, strategies both support autonomous and safe mobile agents).

**Benefits to Mobile Agent Users:** Mobile agent user is free to disconnect from the network, whilst having a high assurance that its agent is autonomous and its mission results will be protected.

**Benefits to Mobile Agent Platform Owners:** Since users will feel safer deploying agents to agent platforms, mobile agent platform owners will be pleased that their platform services are used more often.

**Known Existing Similar Schemes:** There are no mobile agent solutions which provide a reliable and simple means for generic mobile agent tasks to be completed in total remotely. Roth’s (and similar schemes proposed by others) mutual protection of cooperating agents proposal [164] (discussed in Section 4.2.4.1) rely on trusted agent platforms in mutual cooperating agent itineraries, but appropriate itineraries may either be difficult to ascertain or trusted agent platforms may not be available. In any case, those trusted platforms in the itineraries are not TTPs per se since they only assist agent users.

### 5.2.4 Secure Payment

**Problem Statement:** How can agents perform e-commerce transactions securely?

**Proposed TTP Security Service:** Secure E-commerce Transaction Hub Service.

**Benefits to Mobile Agent Users:** Can significantly reduce credit fraud by
5.2. Utilising TTPs to Solve Disparate Problems Between the Major Stakeholders

only performing e-commerce transactions on the TTP secure hub. The agent’s credit payment details are not carried with the agent as it performs work on agent platforms.

**Benefits to Mobile Agent Platform Owners:** Makes the payment process simpler for all, and it is easier to track credit fraud.

**Known Existing Similar Schemes:** No such generic TTP secure hub solutions.

The TPE proposal (discussed in Section 5.1.3) suggests building tamper-resistant agent platforms produced by a TTP hardware manufacturer. Karjoth’s protocols in making use of tamper-resistant devices for (some) merchant operations [110] (discussed in Section 5.1.4) also suggest they could be employed for these tasks.

5.2.5 Secure Inter-Agent Messaging

**Problem Statement:** How can an agent securely message a cooperating (possibly remotely located) agent?

**Proposed TTP Security Service:** Inter-agent Secure Messaging Hub Service.

**Benefits to Mobile Agent Users:** No message snooping by agent platforms is possible. Moreover, there is no need for an agent to carry a secure messaging key with it to mobile agent platforms.

**Benefits to Mobile Agent Platform Owners:** Cannot be accused of snooping secured inter-agent communications.

**Known Existing Similar Schemes:** None.

5.2.6 Access Control

**Problem Statement:** How can mobile agent access control be managed and roles distributed?

**Proposed TTP Security Service:** Centralised Access Control Role Management and Attribute Authority Service (see Section 6.3.3).

**Benefits to Mobile Agent Users:** The mobile agent user does not have to apply to each platform for gaining access permissions for agents to work on that platform.

**Benefits to Mobile Agent Platform Owners:** Can concentrate on delivering quality services, and not have to worry about managing user access control rights assignment and revocation.
Known Existing Similar Proposals: We discussed Jansen’s privilege management scheme for mobile agents [95] in Section 5.1.1. We also touched on SOMA’s domain-based roles [43] concept in Section 5.1.2.

5.2.7 Agent Itinerary Management

Problem Statement: How can an agent’s itinerary be monitored and dynamically changed, and its state checked for safety?

Proposed TTP Security Service: Itinerary Management Service (see Section 6.5 and Section 8.3).

Benefits to Mobile Agent Users: The agent is autonomous of its user, and capable of dynamic update changes (calculated on the secure hub) to its itinerary.

Benefits to Mobile Agent Platform Owners: Agents can be scanned for viruses and common logic attacks by the TTP secure hub, adding greater assurance for mobile agent platforms that agents are safe to do work on their platforms.

Known Existing Similar Proposals: The Supervisor-Worker Framework (discussed in Section 4.2.4.8) allows dynamic updates to the itinerary (but assumes that the master agent in its model is located at a trusted platform - this may not always be possible). In [228], by using the tamper-resistant TPE, it is shown how an agent’s itinerary can be protected.

5.2.8 Resource-Hogging

Problem Statement: How can resource-wasting and zombie agents be efficiently detected and zapped, thus helping to trap denial-of-service attacks from both rogue agents and agent platforms before significant damage is imparted?

Proposed TTP Security Service: Time-to-Live (TTL) Monitoring Service (see Section 6.6).

Benefits to Mobile Agent Users: Agents will be invalidated and recalled if they experience greater than reasonable delays at any agent platform. To track an agent’s present location, the TTP is sent a message just before the agent migrates to its next platform.

Benefits to Mobile Agent Platform Owners: Resource hogging agents will be retracted, and unreasonable time delays in servicing of agents can be tracked. Known Existing Similar Schemes: No such generic TTP security service, to the best of our knowledge, has been proposed. Other denial-of-service detection
mechanisms exist (e.g. the investigation and trust building mechanism [45] as discussed in Section 4.3.3) but they are aimed at only protecting one stakeholder, and often are retrospective mechanisms (e.g. the investigation and trust building mechanism - because it does not involve a TTP - only works post-mission, is only triggered on agent user suspicion, and violates the principle of autonomous and secure execution which is the ultimate goal for mobile agents).

5.2.9 Agent Location Tracking

**Problem Statement:** How do users/agents locate useful agent platforms to assist in their work mission, and monitor the location of deployed agents?

**Proposed TTP Security Service:** Equitable Name-Lookup And Location Service (see Section 6.4 and Section 6.6).

**Benefits to Mobile Agent Users:** Agents can easily find platforms which will assist in completing their mission well.

**Benefits to Mobile Agent Platform Owners:** Agents will more readily use the services of agent platforms.

**Known Existing Similar Schemes:** SeMoA’s Atlas service which supports privacy of mobile agent users where possible [165, 170]; does not provide an equitable name-lookup service - just secure agent location tracking.

5.2.10 Key-Management

**Problem Statement:** How do agents locate the necessary platform public keys so they can transport themselves securely to those hosts?

**Proposed TTP Security Service:** Key Management Service (transparently via proxying secure hub, and mutual authentication service - see Section 6.4).

**Benefits to Mobile Agent Users:** Do not have to know or retrieve agent platform public keys for securely transporting agents there.

**Benefits to Mobile Agent Platform Owners:** Because key management is centralised, secure access to platform services is readily available.

**Known Existing Similar Schemes:** Any security-focused mobile agent system (e.g. like SOMA [43] or SeMoA [169]) has agent platform credential look-up services but often these are agent user initiated queries and not capable of supporting in-mission agent platform credential look-up querying from an agent to a TTP.
5.2.11 Accountability

**Problem Statement:** How can malicious agents and malicious agent platforms be detected, and the responsible entities be punished?

**Proposed TTP Security Service:** Central Auditing Service (see MASHIn RECDAM strategy in Section 8.3 and MASHIn confidence scores framework in Chapter 9).

**Benefits to Mobile Agent Users:** Malicious mobile agent platforms can be detected and punished according to a central, uniform policy.

**Benefits to Mobile Agent Platform Owners:** Malicious mobile agents (and, in particular, the agent’s user) can be detected and punished according to a central, uniform policy.

**Known Existing Similar Schemes:** Hohl’s reference states [86] and Vigna’s execution tracing [212] mechanisms (discussed respectively in Sections 4.2.2.2 and 4.2.2.1) - both only work to detect malicious agent platforms, with the latter mechanism only working post-mission completion and triggered on the suspicions of a mobile agent user. Moreover, neither mechanism offers a means of enabling punishment for malicious behaviour, only some attack detection (e.g. of malicious modifications) capabilities.

5.2.12 Finding Reputable Stakeholders

**Problem Statement:** Which agent platforms have a solid track history of security and customer satisfaction? And, which agent users have a solid track history of deploying authentic and virus/malicious-logic free agents?

**Proposed TTP Security Service:** Major Stakeholder Reputation Service (see MASHIn confidence scores framework in Chapter 9).

**Benefits to Mobile Agent Users:** Has more confidence in sending agents to reputably good mobile agent platforms.

**Benefits to Mobile Agent Platform Owners:** The more well-behaved an agent platform is, the better its reputation is recorded and published, meaning it will be a more attractive destination for agent migration and use. Moreover, it will be more confident in receiving virus-free, non-malicious agents.

**Known Existing Similar Schemes:** None (which assist both stakeholders).
5.2. Utilising TTPs to Solve Disparate Problems Between the Major Stakeholders

5.2.13 Mobile Agent TTP Security Services Summary

We have observed that Internet mobile agent security problems are generally not well-handled by existing TTP solutions, and even when so-called TTPs have been utilised they are in fact often not what we call true TTPs because they seem biased in offering strong protection guarantees for only one stakeholder. In devising our infrastructure-oriented strategy, we offer TTP-based innovations which (in some areas) improve on existing proposed strategies to address the mobile agent infrastructural security problems outlined above. Moreover, we unite several approaches into a more comprehensive infrastructural solution, instead of offering piecemeal mitigation to only one or two problems. Besides this broader resolution coverage of problems (and services offered) at an infrastructural level, the standout distinction we see in our work from previous work is that we utilise TTPs genuinely to (assist both stakeholders and) bring the two major stakeholders closer together so that working relationships can be initiated.

Dedicated trusted third party services are important because they promote scalability of mobile agent services and assist in mitigating the mutual mistrust concerns between mobile agent users and platform service providers. Consider the general Internet (i.e. untrusted inter-domain) mobile agent scenario. The proliferation of interaction among untrusting mobile agent parties in this environment cannot eventuate without the introduction of stronger safeguards for both sets of parties.

We relate the mobile agent stakeholder mutual mistrust complexity to the problem of fair exchange of goods between two parties in electronic transactions. Pagnia and Gartner’s (paraphrased) comments on that subject were as follows: *It is not the question whether a peer is cheating which seems most salient, but rather how to guarantee progress when each participant may fear that its next step will be a losing one. Overall, the notion of adding a trusted third party to a system seems to be the equivalent of introducing some explicit or implicit form of synchronous accord stimulating headway.*

We see this well-articulated assessment correlating well with the paradoxical interests limiting the mass deployment of mobile agents on the Internet, driving an electronic services marketplace - their great original marketing promise. However, from the overview presented above, we do not see evidence of broad generic TTP security service coverage for bringing mobile agent users and mobile agent platform owners into an environment conducive to interaction and the initiation
of useful work. In the next section (i.e. Section 5.3) we introduce the TTP components integral to our secure strategy for enabling Internet mobile agents.

5.3 TTP Components in Proposed Security Infrastructure

We have conceptualised a secure, infrastructure-oriented strategy to bring the major Internet mobile agent stakeholders closer together and to promote the initiation of working relationships between them. Our infrastructure-oriented strategic concept is titled the Mobile Agent Secure Hub Infrastructure, or MASHIn for short.

In the following subsections we provide a brief overview of the TTPs integral to realising our MASHIn strategic objectives. Their roles in the infrastructure are only briefly discussed here; in-depth details can be found in subsequent chapters of the thesis, starting with a much broader introduction to the MASHIn concept in the next chapter (i.e. Chapter 6). The TTP infrastructural components tailored and incorporated for utilisation in the MASHIn concept - registration authorities, certification authorities, attribute authorities, Mobile Agent Secure Hubs (MASHs), and checking hosts - are briefly outlined below.

5.3.1 MASHIn Registration Authorities

All MASHIn entities - including agent platform owners, agent users, MASHs (see Section 5.3.4), and checking hosts (see Section 5.3.5) - must register with MASHIn-accredited Registration Authorities (RAs) before they are allowed to participate in the infrastructure.

RAs are authorities in a distributed network, such as the Internet, that verify user requests (specifically by checking identity credentials) for digital certificates and informs a Certification Authority (CA) to issue it. RAs are part of a public key infrastructure (PKI), a well-defined and genuinely well-understood concept implemented in many traditional distributed systems. PKI has many associated standards and is a widely resourced subject [3, 145].
5.3.2 MASHIn Certificate Authorities

MASHIn entities are issued public key certificates by MASHIn-accredited Certificate Authorities (CAs). Dating from 1976, public key cryptography (sometimes referred to as asymmetric cryptography) [163, 52], allows a more scalable means of securely communicating with distributed entities, especially as that distributed communicating population grows considerably in size.

Public key certificates can be thought of as a digital passport that serves as proof of identity. It is issued by a trusted authority, the CA, and provides identification for the bearer. The CA is the very foundation of PKI since it is the only component entity that can issue public key certificates. MASHIn CAs are responsible for managing the life-cycle of MASHIn entity public key certificates - including creation, issuing, dissemination, revocation, and renewal [89, 3].

5.3.3 MASHIn Attribute Authorities

Mobile agent users apply to MASHIn-accredited Attribute Authorities (AAs) for role-based credentials encapsulated in attribute certificates. The role-based credentials are privileges granted for logical access to agent platform resources, lowering the access control management load for agent platforms. MASHIn role-based access control via attribute certificate issuing and verification is directly based on Jansen and Karygiannis’s privilege management scheme for mobile agents [97] (discussed in Section 5.1.1).

Privilege Management Infrastructures (PMIs) - a different concept and function of PKIs, but with some similar named/analogous entities - are widely seen as a promising means of handling access control functions (including privilege discovery, assignment, dissemination, delegation, and enforcement) in distributed systems [31, 32, 29, 97]. The suggested functions of a PMI are increasingly being standardised [90, 30].

5.3.4 MASHIn Mobile Agent Secure Hubs

Mobile Agent Secure Hubs (MASHs) are the central and most influential TTP abstraction in the MASHIn concept. MASHs are the lynch-pin component in our secure strategy for bringing the two mobile agent system stakeholders (i.e. mobile agent users and mobile agent platform owners) into an environment more
conducive to interaction and the formation of long-standing working business relationships.

We foresee wide security responsibilities for MASHs including the services of authentication, authorisation, anonimisation, agent location tracking, transaction non-repudiation, event monitoring, and trust facilitation - all on behalf of both major mobile agent system stakeholders.

In reality, MASHs may well use the TTP services offered by other specialist security providers such as RAs, CAs, AAs, time-stamping servers, secure billing companies, and so on. Reasons for the suggested distributed deployment of MASHIn security services, offered by many providers, are detailed in the next chapter (i.e. Chapter 6).

MASHs are responsible for coordinating and inserting checking hosts (introduced in Section 5.3.5) appropriately into the mobile agent’s itinerary for assisting in monitoring mobile agent execution integrity and non-repudiation of mobile agent stakeholder inputs.

### 5.3.5 MASHIn Checking Hosts

MASHIn Checking Hosts are MASH-delegated TTP agent platforms inserted in the mobile agent’s itinerary between untrusted agent platforms. Their purpose is to verify the reference states claims from the previous agent platform, and if checked correctly to forward the agent onto the next agent platform in its itinerary.

The results from the checking host’s reference states verification are pivotal to our reputation scheme for the MASHIn (discussed in detail in Chapter 9) which employs first-hand TTP observations as a reliable means of ensuring mobile agent system entities with a track record of sound behaviour are recommended first to those searching for other stakeholders to interact with.

Checking hosts, in unison with MASHs and other checking hosts, also play a key role in helping alleviate denial-of-service attacks and/or non-malicious delays in mobile agent processing by agent platforms, adding robustness to the mobile agent’s mission and the continuity of agent platform processing capabilities. Moreover, checking hosts reliably assist in keeping the location of a mobile agent known at all times. These MASHIn fundamental security services are discussed

---

4 We apply Hohl’s reference states scheme [86], first discussed in Section 4.2.2.2, for our own means - elaborated on in-depth with our proposed utilisation in Chapter 8.
5.4. Summary

This chapter focused on the utilisation of Trusted Third Parties (TTPs) in assisting to secure Internet mobile agent applications and/or the Internet mobile agent model per se.

In Section 5.1 we provided an overview of how some mobile agent applications, or some schemes dealing with securing parts of their mobile agent systems (model), have incorporated TTPs in their security strategies. Some of these schemes have appropriately adopted more traditional distributed system TTP services such as certification authorities, domain servers, and attribute authorities for making system-specific or application-specific security service provision more credible and scalable.

In Section 5.2 we presented a number of security problems, from a global infrastructural perspective of potentially thousands of agent platforms and millions of agents, that we see as a great limitation contributing to the state of mutual mistrust between the mobile agent system stakeholders - thus limiting the feasibility of wide adoption of agents working securely in a dynamic Internet mobile agent community.

We took a first look at the TTP components integral to our secure strategy, the Mobile Agent Secure Hub Infrastructure (MASHIn), for enabling secure usage of Internet mobile agents in Section 5.3. Some of these MASHIn TTP components (RAs, CAs, and AAs) have been applied extensively in more traditional distributed systems and for mobile agent system-specific security provision. We apply them at an infrastructural level, incorporating them with our other MASHIn TTPs (MASHs and checking hosts) to lower the concern levels of both mobile agent system stakeholders, bringing the two stakeholders into an environment more conducive to interaction and long-standing business relationships.

The thesis remainder expands in-depth on our MASHIn strategy. Chapter 6 provides a more thorough overview of the MASHIn concept, component entity interaction, and how we conceptualise addressing some of the largely unsolved
infrastructural problems detailed in Section 5.2. In Chapter 7 we discuss our approach to achieving mobile agent code and data privacy within the MASHIn framework; we will also discuss how the lesser problem of addressing mobile agent platform privacy can also be accommodated in the MASHIn. Our execution integrity approach is elaborated on in Chapter 8, with outputs from that process being partial inputs to the MASHIn reputation scheme, discussed in Chapter 9 for bringing previously unknown (in identity and trust dependability) mobile agent stakeholders closer together.
Chapter 6

Foundations of the Mobile Agent Secure Hub Infrastructure

A person who trusts no one can’t be trusted.

Jerome Blattner

In the last chapter we commenced our discussion on how TTPs can be an effective means of bringing the two previously mistrusting mobile agent system major stakeholders closer together into an environment conducive to business interaction. In Section 5.3 we briefly touched on the TTP component entities we utilise in the Mobile Agent Secure Hub Infrastructure (MASHIn), our secure strategy for enabling Internet mobile agents.

In this chapter we significantly elaborate on the conceptual strategy we have devised for the MASHIn. In particular we expand on the roles played by the TTP components employed in the MASHIn and their interactions. Where possible, we provide a number of diagrams to better illustrate the pertinent work-flows.

We commence in Section 6.1 by reviewing the overriding goal of the MASHIn - that of bringing the two major stakeholders into an environment conducive to interaction by lowering their fear that their next step may be a losing one. We also stipulate sub-category objectives which we strive to achieve in working towards this main goal.

In subsequent sections, we take a look at the logical steps taken by the major
stakeholders and entity interactions when utilising incorporated MASHIn TTPs. In Section 6.2 we discuss the MASHIn registration and credential issuing processes. Scalable infrastructural access control is discussed when we outline the MASHIn privilege management scheme and processes in Section 6.3. The authentication and anonymisation processes of the MASHIn are discussed in Section 6.4. Finally, in Section 6.5 we look at how the location of an agent can be autonomously monitored and delays in processing - whether from intentional malevolent behaviour such as denial-of-service attacks, or agent platforms experiencing bottleneck delays - can be detected and relieved in real-time (i.e. during the agent’s mission).

This chapter includes and builds on material found in peer-reviewed accepted papers [i]-[iii] listed on page xxi.

## 6.1 MASHIn Impetus and Strategy

From the conclusions reached in the previous chapter on reviewing existing mobile agent security strategies utilising TTPs, we observed that often the incorporation of TTPs has been done so at a mobile agent system-level (i.e. localised single-agent-system environment) rather than infrastructural-level (i.e. multi-domain, multi-agent system capable of supporting large numbers of agent users and agent platforms in an application-independent environment). Moreover, the TTPs in those strategies were regularly serving to reduce the risk concerns of only one major stakeholder. These shortcomings are not only detrimental to the interoperability of incorporated TTP mechanisms, but do nothing to bring the two mistrusting stakeholders closer together (which is a clear problem needing prompt addressing for the prospects of Internet mobile agents to move forward).

It is impractical to investigate either protecting agents or protecting agent platforms in isolation of the other. Both sets of parties (i.e. agent users and agent platform owners respectively) must have compatible, or at least accommodating, security policies and precautionary measures to ensure a long and fruitful business relationship. In devising our strategy, our work would focus at an infrastructural (rather than system) level, and TTPs would be utilised for the benefit of both stakeholders.
6.1.1 Design Goals for the MASHIn

We were particularly mindful of a number of objectives in the design goals for the MASHIn, with the rationale for the objectives, as follows:

- *The sensitive operations of a mobile agent should be kept totally inaccessible to untrusted mobile agent platforms:* Agent platforms should not be able to subvert agent code to utilise sensitive operations inappropriately, and they should learn less of the mobile agent’s susceptibilities/thresholds.

- *The problems in discovering appropriate business partners for mobile agent users and agent platform owners should be eased, and consequently a reduction in the key management problems for both sets of parties should be realised:* This would assist the establishment and growth of a dynamic mobile agent community able to conduct business in a more secure fashion.

- *Mobile agent execution integrity must be maintained:* Agents will be interpreted as they are intended, and thus agent users will only be held accountable for their agent’s correct interpretation. Agent platforms can also be held accountable for their inputs to the process.

- *The integrity of mobile agent data needs to be protected and mobile agent data values must be non-repudiable:* Agents (and subsequent agent platforms) will therefore be less susceptible to being coerced into performing an action under a false assumption.

- *Misbehaving entities should have their reputation lowered:* The punishment is a strong deterrent against future malevolent actions.

- *The identity of a mobile agent user, if desired, should not be disclosed to agent platforms:* Mobile agent users are less susceptible to being spammed by agent platforms that their agents visit, nor can a profile be surreptitiously built on the mobile agent user.

- *The itinerary of a mobile agent, if desired, should be kept secret from agent platforms:* Mobile agent platforms learn less of the agent’s purpose, and are in a weaker position to initiate collaboration attacks (with other agent platforms in an agent’s itinerary) against mobile agents.
6.1.2 Basic Purpose of the MASHIn

The Mobile Agent Secure Hub Infrastructure (MASHIn) is our high level conceptual community of Internet mobile agent platforms and Internet mobile agent users who are dually interested in safe interaction and accountability for breaches of security policy. The central abstraction in the MASHIn are Mobile Agent Secure Hubs (MASHs). MASHs act as unbiased mediators and hold both parties accountable for their actions.

The MASHIn is a novel conceptual offering for raising the level of confidence in a mobile agent’s results and safety. A mobile agent user sends the Mobile Agent Secure Hub (MASH) a message asking for mission protection of its mobile agent. Actually, this is a simplification of the process, as we will discuss in Chapter 10, because the agent user should not directly be able to communicate or choose any one specific MASH. This helps to promote TTP neutrality, and mitigates the chances that MASHs are a visible network target for denial-of-service or other attacks. However, for simplifying a number of examples that we present up until that chapter, we do not show this non-traceable indirection between agent user and the MASH assigned mission protection of the user’s agent.

During the lifetime of the agent’s mission, the MASH is delegated the responsibility of managing and monitoring the itinerary and safe execution of the mobile agent.

As a trusted third party, the MASH reduces the mutual mistrust lingering between mobile agent users and mobile agent platform service providers and prescribes security measures accommodating both sets of parties.

It is important to note that the MASH may well use TTP services offered by other specialist mobile agent security providers (for more discussion see Section 6.1.4) in fulfilling its responsibilities, but to keep things conceptually easy to follow we refer to the TTP entity providing mobile agent security services simply as the MASH.

6.1.3 MASHIn Core Components

Figure 6.1 depicts the major software modules we foresee for MASHs. The vertical layering gives some insight into the dependencies of modules, with higher stacked modules building on the services delivered by lower modules.

The Authentication Module is responsible for authenticating the own-
ers of incoming agents requesting mission protection, and passing authenticated agents onto the authorisation module. Authentication of agent platforms and agents cooperating via secure inter-agent messaging is also the responsibility of the authentication module. MASHIn authentication is discussed more in Section 6.4.

The **Authorisation Module** is responsible for verifying an agent owner has been granted a claimed role, or role set, for its agent’s work on target agent platforms. This module also acts as a helper to other modules needing access control determinations - for example, it may assist the agent location tracking module in determining whether a requesting entity is allowed to know an agent’s location. Considering another example, the authorisation module will also assist the transaction non-repudiation module in determining which agents can participate in business transactions, and under what conditions. Section 6.3 discusses the processes associated with the MASHIn’s access control model and determining authorisations for a principal via role-based privilege management.

The **Anonimisation Module** is responsible for generating and keeping specific instances of authorised user/role anonimisation mappings secret from agent
platforms. Higher level modules will interface with and through the anonymisation module to determine access control rights and record non-repudiable transactions involving mobile agents. MASHIn anonymisation is discussed more in Section 6.4.

The **Agent Location Tracking Module** is responsible for receiving mobile agent location updates; as a prerequisite sub-service it would formulate unique identifiers for agent instances in the MASHIn. MASHIn agent location and tracking is discussed more in Section 6.5.

The **Transaction Non-repudiation Module** is responsible for storing a non-repudiable record of mobile agent e-commerce related transactions, and charging both stakeholders for utilisation of MASHIn TTP security services. It has the added responsibility to ensure non-disclosure of stakeholder payment details. A mechanism for non-repudiation of stakeholder inputs in ensuring mobile agent execution integrity in the MASHIn is discussed in Chapter 8, as well as a guide to a fee charge for stakeholders utilising this MASHIn security service.

The **Event Monitoring Module** monitors a wide variety of events within the MASHIn - for example agent migrations, agent state checking failures, and bottlenecked agent platforms. Relevant MASHIn event monitoring tasks are discussed more in Section 6.5 and Chapters 8 and 9.

The **Trust Facilitation Module** heads the tiered group of software models for MASHs, after all the MASH’s core function is acting as a trusted third party between mobile agent users (more specifically their mobile agents) and mobile agent platform service providers (more specifically their agent platforms). Trust relationships can be statically defined, based on transitive entity relationships, or dynamically updated in response to events like an agent platform reaching some unacceptable threshold of mobile agent reference state failures (discussed in Chapter 8). Moreover, as discussed in Chapter 9, MASHs can facilitate the formation of personalised entity confidence score appraisals.

### 6.1.4 MASHIn Deployment Considerations

We stipulate, once more, that we do not expect a single server to be responsible for offering all expected MASHIn services. Deployment of MASHIn security services should be distributed and offered by many providers, for a number of reasons including:

- No MASHIn TTP should be the equivalent of a supreme power. That is, relying solely on any one entity gives that entity too much power. If
that entity is attacked (for example, via a denial of service attack), or it is compromised (either via coercive ‘soft’ tactics or crafty hacking), the TTP security concept is broken because there is only a single point-of-failure.

- De-centralised TTP location and management is important so the transport and processing penalties of TTP utilisation are minimised wherever possible.

- The MASHIn should be capable of serving a large number of mobile agents (possibly millions at any one time).

- TTP data backup and data replication procedures would be required - both necessitate a distributed system.

- There will be entities who only trust certain TTP companies or hardware configurations to capably perform some TTP security services, whilst dismissing their value at offering other TTP services - for example, a mobile agent user may trust a TTP with authentication and anonymisation services, but not with mobile agent execution integrity checks and non-repudiation of agent inputs.

- We anticipate that some TTPs would serve particular jurisdiction and/or locale boundaries.

Considering the major software security services depicted in Figure 6.1, we anticipate close coupling (perhaps in-organisation housing) between the Authentication, Authorisation, and Anonimisation modules. The Agent Location Tracking responsibility would be processor intensive\(^1\), and we anticipate it would be the function of servers dedicated solely to that purpose. The Event Monitoring module would involve stationary agents near the source of events communicating with appropriately distributed handlers (case specific to the event/action). Dedicated servers should handle the Transaction Non-Repudiation module functions, because these functions are critical to ensuring safe stakeholder interaction and also include billing stakeholders for utilising MASHIn security services (thus they need to be implemented and managed in highly-trusted environments having

\(^1\)The MASHIn should be capable of handling possibly millions of agents and thousands of agent platforms. Thus, with this number of entities interacting, agent location message updates as agents move from platform-to-platform, and cooperating entities querying the location of agents this module would be heavily processor intensive.
clear separation-of-concerns). And, yet again, we envisage dedicated entities to handle the responsibilities handled by the Trust Facilitation module as this module manages sensitive stakeholder trust relationships and stakeholder reputation data inputs needing strong integrity protection and regular updating (discussed in Chapter 9).

Other deployment variations could be seen. We do not wish to be over-committed or limiting in stipulating required infrastructural entities or inter-entity relationships at this stage of our MASHIn concept proposal.

6.2 MASHIn Entity Registration Processes

A principal is an entity that has a provable identity. In the MASHIn, the principals of most interest to us are the major mobile agent paradigm stakeholders (i.e. agent users and agent platform owners). The ability to authenticate, or prove, the identity which a principal may claim (see Section 3.1.1) is the key security function on which all other security functions (e.g. integrity and non-repudiation) rely in a distributed system.

Asymmetric key (or public key) cryptosystems are generally more scalable than symmetric key cryptosystems, not only because the total number of entity keys required for secure communication is dramatically reduced\(^2\) but there is also a reduced likelihood for a real-time key-distribution service (such as that provided by the Key Distribution Centre component in Kerberos [11, 136]). They also tend to be better suited to situations where previously unrelated principals need to interact securely - a common scenario in a large-scale, dynamic mobile agent infrastructure such as the MASHIn strategy. Instead of a real-time key-distribution service, each principal in asymmetric key cryptosystems makes their public key generally available (for example on a personal user’s web page or in some publicly-accessible repository such as a Lightweight Directory Access Protocol LDAP [218] server).

One of the main problems that a PKI aims to solve is the issue of how does one principal know for certain that a public key belongs to a principal making such a claim. This is the role of a Certification Authority (CA), a TTP, who

\(^2\) Each entity only needs a private and public key in asymmetric key cryptography (i.e. \(2 \times n\) entities keys in total for the community) to securely communicate, compared to a shared-key for \(n\) entities each entity has \(n\) keys (i.e. \(n^2\) keys in total for the community) to securely communicate employing symmetric key cryptography.
verifies the identity of a principal before digitally signing their public key certificate. Any principal, who has access to the CA’s public key corresponding to the certificate-signing key is able to verify the authenticity and integrity of a public key certificate signed by that CA. It is possible for a principal to verify the authenticity and integrity of a public key certificate that was signed by another CA, often via a hierarchical distributed trust architecture (like the X.509 PKI strategy [3, 90]) or ad-hoc user-centric trust (like the PGP PKI strategy [66, 3]) model relationship. A recent non-adhoc distributed trust framework proposed by Burmester and Desmedt [26] attempts to reduce some of the vulnerabilities of a hierarchical PKI by allowing a principal to verify the authenticity and integrity of a public key certificate based on a voting majority of horizontal trust-graph path certifying entities.

The notion of registration, including establishment and confirmation of a principal’s identity, directly implemented by a CA is problematic in a centralised form—especially as the number of end-entities in a given PKI domain increases and/or the end-entities are geographically widely dispersed. Thus, it makes sense to off-load the registration function to a separate TTP entity known as a Registration Authority (RA). The process by which an RA verifies the identity of a requesting principal seeking public key certification is detailed in a Certification Practice Statement (CPS) document [90, 3].

Multiple trusted RAs, judiciously deployed and sometimes referred to as Local Registration Authorities (LRAs), can enhance scalability of a PKI and decrease operational costs for CAs [3]. An RA generally provides the certificate management interface for users of the PKI, enabling a CA to work off-line. It can be mandated that only strongly authenticated RAs may contact a CA, reducing some of the risks associated with allowing general access to this crucial infrastructural-security component.

Based on the documented CPS procedures which are in use, the RA accepts requests for certificates on behalf of a CA and verifies the binding between the public-private key-pair and the identity of the requesting principal. Once the binding has been verified, the principal’s certificate request is securely forwarded on to the CA, and the CA will issue a certificate based on the RA’s recommendation. The RA can take on a varying number of roles including [3]:

- Initiating the certification process with a CA on behalf of individual end-users (including the registration of certain attributes to be associated with
Chapter 6. Foundations of the Mobile Agent Secure Hub Infrastructure

the end-user),

- Generate keying material for a CA (indirectly on behalf of the end-user), and

- Perform certain key/certificate life cycle management functions, such as to initiate a revocation request or a key recovery operation for a CA (working on behalf of the end-user).

With respect to the registration concepts just outlined, MASHIn entities must register with a valid MASHIn-accredited RA (not a CA). By valid, we mean the employed RA must have and oblige by a certified MASHIn-aligned CPS.

MASHIn CAs, in interfacing with MASHIn-accredited RAs, certify the binding of a stakeholder’s identity (and potentially other attributes like validity periods, contact information, etc.) with a public key via the encapsulation of this information in a signed data structure known as a public key certificate. Public key certificates are digitally signed with the private key of the MASHIn-accredited issuing CA. Since the CA digitally signs the certificates, they are self-protected from an attack against their integrity and thus can be freely disseminated safely - assuming they do not contain any sensitive information. Public key certificates may be “pulled” from repositories such as LDAP servers and corporate databases. Authenticated access to these repositories can be applied if sensitive information should only be accessible to privileged principals [3].

In the MASHIn context certificates should be retrieved from dedicated repositories to minimise load on any one entity. It would make sense to replicate this information locally at MASHs since they are the pivotal entity assisting the two major MASHIn stakeholders and need to readily be aware of the public keying details of both sets of stakeholders.

MASHIn CAs are responsible, in partnership with CRL Issuers, for issuing Certificate Revocation Lists (CRLs) for revoking public key certificates in a timely manner (i.e. cancelling a given certificate before it would naturally expire) possibly triggered by any number of events such as a suspected private key compromise or a principal’s change in job status, location, or contact details.

Figure 6.2 shows the MASHIn RA and CA roles, and the interaction between these two TTPs and major MASHIn stakeholders (i.e. agent users and agent platform owners), many of which would be conducted off-line (such as LDAP certificate retrieval and batch certificate management processing tasks such as
key/certificate updating). It is pertinent to note that the model includes flexibility, or provision, for more than one of each entity (e.g. RA, CA, etc.), but the roles of each entity remain unchanged regardless of the number of like-participating entities within the MASHIn.

![Diagram showing MASHIn RA, CA and major stakeholder responsibilities and interactions](image)

**Figure 6.2:** MASHIn RA, CA and major stakeholder responsibilities and interactions (adapted from [113]).

Whilst the CA is the very foundation of the MASHIn PKI since it is the only entity that can issue public key certificates, the RA component assumes a number of administrative functions from the CA - taking complete responsibility for stakeholder registration and possibly offering assistance in a number of other areas as well. Both sets of stakeholders (i.e. agent users and agent platform owners) require the initial generation of a secure public key-pair (to communicate with the other stakeholders, and TTP components like MASHs and checking hosts) and periodic long-term public key-pair updating. Certificates needing revocation must also be processed and attributed as such by CAs or a cooperating TTP entity, the *CRL Issuer*. The repository is a generic term for the various data containers (LDAP servers, DNS servers, OCSP servers, FTP servers, etc.) holding a copy of certified certificates and CRLs that stakeholders can retrieve on-demand.
as necessary [113].

While traditional PKI is a relatively non-ambiguous technical concept and widely incorporated in various forms for beefing up security in homogeneous communities, there are a number of imitations and shortcomings with PKIs which have slowed their uptake. We realise that these problems would also be of concern in designing, and effectively utilising, a PKI within the MASHIn. Specifically these problems include, but are not limited to:

- A number of steps and links are involved in PKIs, including RAs and CAs and their various process responsibilities. The large number of operations constituting a PKI may create vulnerabilities because they involve both cryptographic operations and people interfacing in a number of processes. A large-scale chain of operations and processes can be thought of as being only as strong as its weakest link.

- The financial benefits of employing PKIs are misunderstood. PKI refers to the infrastructure and processes necessary to enable secure applications, it does not deliver any immediate value to an entity’s bottom-line. This lack of a direct tangible output should not, nevertheless, be a deterrent against investment in PKI technologies (i.e. for their value in protecting applications, and most importantly the data of value to people in those applications).

- We are yet to see practical evidence of global, cross-community PKIs of any considerable scale. Interoperability issues (of both a technical and political nature) pose a significant limitation, as does the business issue of who remunerates who in a large-scale PKI.

- Which CA is trustworthy? Besides entrusting in a brand-name or reputable organisation, what non-anecdotal evidence is available to justify the supposed credible practices of any CA or chain of CAs?

- The process of revoking certificates, retiring asymmetric cryptographic key-pairs, and attaining new key-pairs/certificates has yet to find a robust solution. This is because a wide array of applications and processes all need to follow correct procedures\(^3\) in a timely manner for entities to seamlessly continue to securely communicate over multiple key-pair lifetimes.

\(^3\)Often these are not well-defined or interoperable.
In the next section we build on the base MASHIn PKI model, with TTP components and processes, discussed in this section to accommodate large-scale mobile agent access control to agent platform resources.

6.3 MASHIn Scheme for Authorisation to Agent Platform Resources

In the previous section we looked at and discussed the underlying basic PKI model which forms the foundation for scalable authentication between major stakeholders and TTPs in the MASHIn. In this section we extend our infrastructure-oriented mobile agent security strategy by discussing the incorporation of a credential infrastructure component for enabling scalable mobile agent authorisation to agent platform resources. In Section 6.3.1 we discuss candidate access control models and why we chose the role-based access control model as most appropriate for enabling scalable distributed authorisation management in the MASHIn. In Section 6.3.2 we point out existing generic distributed role-based access control schemes, and in Section 6.3.3 we add a credential infrastructure component to the MASHIn model which works in nicely with the PKI component discussed previously.

6.3.1 Candidate System Access Control Model Mechanisms

Authorisation is the act of granting privileges or rights to a principal. Access control is the set of processes employed to ensure that only authorised principals gain access to valued resources. There are a number of access control models including [161]:

- Discretionary access control [12, 153]: Access control decisions are usually based on the identity of the requesting principal. This is a very common model used in modern operating systems for enabling discretionary access to system resources, and it is also used extensively for secure access to web servers on the Internet (via username and password schemes, or public key based schemes). These schemes are simple to use initially in the case of small and infrequently-changing resource environments, but their manage-
ment overheads tend not to adapt well to dynamic, large-scale distributed systems.

- **Mandatory access control** [93, 240]: Access control decisions are based on characteristics or labels which are assigned to resources, and only principals with privileged clearance at least as high as the label attached to a resource may be granted access to that resource. The label-based access control model is used less frequently than the identity-based access control model, but enables stronger access control of resources (via clearances such as top secret, secret, confidential, etc.) and is commonly used in defence force departments. This access control model is generally inflexible for distributed commercial environments, and it is inadequate for the needs of personal information systems.

- **Role-based access control** [161, 142]: Access control decisions are based on the function (via an assigned role) of the principal, rather than solely who a principal is (identity) or what status that have (clearance/label). In *Role-Based Access Control* (RBAC), rights and permissions are assigned to roles rather than to individual users. Users acquire these rights and permissions as they are assigned membership in appropriate roles. This simple idea greatly eases the administration of authorisations. RBAC is a mandatory access control model which has evolved to meet the needs of commercial information systems. Rather than labelling information, RBAC associates roles with each individual who might have a need to access information. Each role defines a specific set of operations that an individual acting in that role may perform.

We chose to employ a RBAC model in the MASHIn for a number of reasons. Firstly, the RBAC model offers the most flexibility and best security management capabilities for distributed environments and large-scale systems. RBAC-modelled schemes have been developed and deployed for enabling authorisations in both generic distributed systems (see Section 6.3.2) and mobile agent systems (as discussed in Sections 5.1.1 and 5.1.2). When utilised in a credential infrastructure (see Section 6.3.3), the RBAC credential infrastructure works well to complement the existing MASHIn PKI. Finally, the abstraction of a role can be utilised to facilitate mobile agent user anonymisation in the MASHIn (see Section 6.4).
6.3.2 Generic Credential Infrastructure Models

A number of mechanisms exist to support the implementation of a credential infrastructure, many of which support RBAC. They fall broadly into three categories [3]:

- Mechanisms which extend the authentication primitives in Kerberos for privilege management purposes (i.e. to add authorisation functionality), including DCE [202, 201] and SESAME [37, 210].

- Mechanisms which employ the concept of a Policy Server - a central server that creates, maintains, and verifies policy for identities, groups, and roles - for example the SOMA (centralised by domain) policy and role servers [43] (discussed in Section 5.1.2).

- Mechanisms based on Attribute Certificates. An attribute certificate is a signed structure similar to a public key certificate, certified instead by an attribute authority rather than a CA, binding some privilege or permission information - possibly via assigned membership roles - to an identity (or perhaps to a public key certificate, which encapsulates that identity). A good example incorporating attribute certificates is the mobile agent privilege management scheme by Jansen and Karygiannis [97] first discussed in Section 5.1.1 and elaborated on with its component incorporation in the MASHIn as discussed in Section 6.3.3. The PERMIS PMI [31, 57, 32] is a good example of a privilege management scheme gaining popularity for its employment of attribute certificates in securing resource access in generic distributed system applications.

As all three access control mechanisms have their advantages and disadvantages, they all have their fair share of proponents and detractors. Kerberos-based schemes tend to be symmetric key-based, and so they have very attractive performance characteristics but their key management and single point of failure characteristics make them non-robust for open Internet environments. Policy server schemes are highly centralised, so they have attractive single-point-of-administration benefits but have unattractive communication overheads and questionable scalability across multiple domains. Attribute Certificate schemes can be fully distributed, and so they have attractive failure resistance but relatively unattractive performance characteristics due to their reliance on public key cryptographic operations [3].
We chose the Attribute Certificate access control model because the MASHIn needs to be capable of supporting potentially millions of agent users and thousands of agent platforms; thus the distributed nature of this model and its inherent scalability was most advantageous over the other two models. The reliance on public key cryptographic operations is a requirement (and not seen by us as a limitation) when considering the secure design of such a large-scale Internet mobile agent infrastructure - in terms of authentication and authorisation scalability, and ease of maintaining non-repudiation of transactions. As we will highlight next, the MASHIn attribute certificate-based credential infrastructure ties in well with the public key certificate-based PKI component of the MASHIn.

### 6.3.3 Credential Infrastructure Component in the MASHIn Model

We propose that MASHIn privilege management to agent platform resources be controlled by RBAC, facilitated by *X.509 Attribute Certificates*.

Public key certificates bind (through CA certification) long-lived characteristics (most importantly a principal’s identity and public key ownership), while attribute certificates can bind (through AA certification) identity with shorter-lived characteristics such as mobile agent user roles. Whilst an attribute certificate is somewhat similar in structure to a public key certificate, they contain no public key. Instead a principal’s identity is usually attainable by “pointers” to the associated public key certificate or matching principal distinguished name/s [3, 90].

There is no requirement that the same authority create both the public key certificate and attribute certificate(s) for a user; in fact separation of duties will frequently dictate otherwise. In environments where different authorities have responsibility for issuing public key and attribute certificates - such as in the MASHIn - the public key certificate(s) issued by a CA and the attribute certificate(s) issued by an AA would be signed using different private signing keys [90].

Besides encapsulating authorised mobile agent roles, mobile agent specific requirements such as *Time-To-Live* (TTL) properties can also be specified in the attribute certificates. Figure 6.3 details the International Telecommunication Union’s (ITU) standardised X.509 Attribute Certificate structure [90]. The X.509 attribute certificate definition aligns closely with the X.509 public key certificate definition, with the content elements relatively self-explanatory.

In terms of a credential infrastructure component for accommodating mo-
6.3. MASHIn Scheme for Authorisation to Agent Platform Resources

Figure 6.3: ITU’s X.509 Attribute Certificate content elements (derived from [90]).

bile agent access to agent platforms in the MASHIn, attribute certificates must be signed by the issuing MASHIn attribute authority to protect the sensitive elements from alteration and to certify to agent platforms that the attribute certificate holder has indeed been granted the claimed roles\(^4\). The Principal (Owner) can be formed by a one-way secure hash over the agent’s code and accompanying information, as suggested by Jansen in [95]. The Attributes element contains the privileged roles granted to the agent that are certified. The optional Extensions element can be employed for definition of new fields such as TTL properties (which would be consulted by TTP MASHs and Checking Hosts performing event monitoring responsibilities of the MASHIn first discussed in Section 6.5).

Though the issuing of roles to agent users and their agents (by the issuing of attribute certificates) is delegated to TTP attribute authorities, the agent platform inherently shoulders responsibility for the processing of policy. This is a reasonable procedure given the design goals for an agent platform with respect to protecting its resources.

Figure 6.4 depicts the various inputs and components needed for MASHIn credential management, from the localised perspective of an agent platform. The encapsulation of agent code and attribute certificates in a *Java ARchive file*

\(^4\)As will become clear in Section 6.4 the agent (user) may be anonymised by MASHs, so the identifying attribute certificate holder will be a MASH who is acting as a proxy delegator for agent anonymisation purposes.
(JAR) may suggest policy containment only appropriate for a Java-based mobile agent system. However, the credential infrastructure scheme is generic to support any agent system environment, or for that matter any set of agent system environments. JAR files are shown in the figure because they are flexible for conveying meta-information such as attribute certificates, can be easily protected for authentication and integrity purposes, and are employed for the packaging of migrating agents by most agent systems due to their utility value and the fact that most agent systems are built on top of a JVM environment.

Policy Certificates encapsulate agent platform-specific\(^5\) trusted policies determined by the pertinent administration team. While an attribute certificate conveys the policy associated with a mobile agent, the policy certificate conveys policy rules governing the behaviour of all agents that may attempt to visit and use agent platform resources [95].

A Policy Engine is the agent platform’s access control enforcement decision component; it interprets the agent platform’s policies (via its policy certificates)

\(^5\)Though, if required, a policy certificate could have domain (i.e. multiple agent platform) scope.
6.3. MASHIn Scheme for Authorisation to Agent Platform Resources

along with certified agent roles (via the agent’s attribute certificates) and their local privilege mappings, to either permit or deny requested access requests by an executing agent.

The procedure in which a MASHIn agent platform determines what privileges are permitted to an (authenticated) agent and enforced at runtime is as follows:

1. Load the agent platform’s policies (usually at start-up time) via the agent platform’s policy certificates.

2. Having been passed the agent’s attribute certificate/s, if and only if they are issued by a trusted MASHIn attribute authority\(^6\), ascertain which encapsulated roles are relevant to the agent’s execution on this agent platform\(^7\).

3. For each relevant role from the above step, determine the local privileges granted to a principal acting under that role\(^8\). This will result in the privilege set of operations allowable for an agent.

4. For every access control request made by the agent, consult the calculated agent’s privilege set (attained in the previous step) and return whether the agent should be granted or denied the access requested to agent platform resources\(^9\).

Our proposed credential infrastructure component for the MASHIn appropriately adapts the privilege management scheme proposed by Jansen [95] for the purposes of our infrastructure-oriented secure mobile agent strategy. By promoting the use of RBAC, with role-related policies encapsulated in attribute certificates, we are increasing the scalability and interoperability of MASHIn-deployed mobile agents.

The MASHIn scales well because agents are:

1. Authenticated by MASHs indirectly through MASHIn CAs, as we discuss next in Section 6.4. This lowers the key management concerns of agent platforms because, to accept agents securely, they now only need to know

\(^6\)In verifying the identity of attribute authorities, the MASHIn PKI may be utilised.

\(^7\)Precedence processing of (principal and role relationships in) multiple attribute certificate may be required.

\(^8\)Some localised role mappings may already be held in the policy engine’s cache.

\(^9\)Enforcement of runtime access control requests is dependent on the underlying agent system security architecture, for example for those systems employing a JVM the agent’s code would be restrained by asserting the associated privilege set of permissions within a protection domain associated to a Java security class loader.
the public key of MASHs (and possibly MASH-delegated checking hosts) instead of potentially thousands of agent platforms.

2. Granted certified roles by TTP MASHIn attribute authorities. This means that a considerable part of the privilege rights negotiation overhead between agents and agent platforms has been taken care of before the agent reaches the agent platform, via the TTP attribute authority issuing of roles to agents, freeing the agent platform to better serve agents. Moreover, changes in local agent platform role-to-rights mappings do not need to be propagated to attribute authorities; this administrative update and enforcement of role-to-rights mappings is localised at each agent platform.

Interoperability is increased because the agent-granted-roles conveyed in attribute certificates are not specifically bound to any one agent system language. This means that the roles issued by attribute authorities to agents are in a generic form capable of interpretation by any MASHIn agent platform base operating system. So, for example, a window shopping agent written in Java and a window shopping agent written in Telescript could be issued a role-set encapsulated identically in an attribute certificate by an attribute authority. Another benefit of generic role-issuing semantics is the ease in which remotely-located agents can permit or deny inter-agent communication with heterogeneous agents (i.e. agents written in different languages), without concerning the agent role’s issuer with the varying specifics of agent platform policy syntax.

### 6.4 MASHIn Authentication and Anonimisation Processes

In this section we provide an overview of the proposed MASHIn authentication and anonimisation processes which lower the key management burden for both agent platforms and agent users.

Figure 6.5 considers the steps instigated by an agent user to deploy their agent in the MASHIn for safe remote autonomous mission completion. These steps are chronologically-ordered but not necessarily immediately triggered one after the other concludes. We now consider each step in turn. The figure depicts agent travel to three untrusted agent platforms; however, this is variable as necessary. Please note well that all messages are sent securely using asymmetric
cryptographic key messaging to ensure message sender authentication, message confidentiality, and message integrity.

**Step 1:** Mobile agent users must register themselves with MASHIn-accredited Registration Authorities (RAs).\(^{10}\)

**Step 2:** The consulted MASHIn RA will, given sufficient and appropriate identification of the agent user, forward this identity information onto a MASHIn-accredited Certificate Authority (CA) for certifying the mobile agent user principal.

**Step 3:** The MASHIn CA will certify the forwarded principal’s identity and the generated public key via a signed public key certificate for the principal which it

---

\(^{10}\)Agent platform (owners) must also initiate this registration process before being allowed to participate legitimately in the MASHIn.
passes back to the client MASHIn RA\[11\].

**Step 4**: The MASHIn RA will pass back the certified public key certificate to the requesting principal. The associated principal’s generated asymmetric private key will be securely shipped back to the requesting principal by an out-of-band mechanism. One possible out-of-band mechanism could be encapsulating the private key on a CD-ROM diskette packaged non-conspicuously and transported via certified mail by a reputable courier to the principal’s residence, with the key on the CD-ROM secured via a time-limited passphrase-protected unlocking key which is telephoned through to the principal (who is asked a series of personal questions for authenticating his/her identity).

**Step 5**: The (now identified) agent user can apply to MASHIn-accredited Attribute Authorities (AAs) for privileged roles so their agents can work as necessary on MASHIn agent platforms.

**Step 6**: If the consulted MASHIn AA permits certain roles for mobile agent user agents, they will return certified attribute certificates (encapsulating the privileged role/s) back to the requesting agent user principal.

**Step 7**: The agent user forwards on their (signed) agent and associated agent attribute certificates to a MASHIn Mobile Agent Secure Hub (MASH) for protected mission execution. As just presented, this scenario step is described in its simplest, least secure case - as MASHIn MASHs may be a target of attempted denial-of-service or active compromise attacks. In Chapter 10 we consider alternative scenario structures and measures for beefing up protection of MASHs - the most-privileged and pivotal TTP in the MASHIn.

**Step 8**: The MASH inspects the incoming (signed) agent package, authenticates the package (and agent user) via checking the accompanying public key certificate against the signed agent package (perhaps performing MASHIn CA certification path verification in the process). If authenticated, the MASHIn proceeds to verify that the claimed agent roles have been issued by a MASHIn-accredited AA. If this also checks correctly, the MASH proceeds to build a proxy attribute certificate with the verified granted roles encapsulated. The MASH then repackages the agent user’s original package contents into a package signed by the MASH, discards the original attribute certificate and instead includes the proxy attribute certificate.

---

\[11\] Either a MASHIn RA or MASHIn CA can generate the principal’s asymmetric cryptographic key-pair, but only a MASHIn CA can certify the associated principal’s public key certificate. There may also be a case for allowing certain principals to generate their own asymmetric key-pair, but that would warrant deeper consideration as it may place unnecessary pressures on enforcing consistency in MASHIn cryptographic key life-cycle procedures.
certificate. Thus, the agent (user) identity has been anonymised by the “prox- ying” MASH, who is trusted by agent platforms (to keep a record of the agent user principal’s details, in the event that some discrepancy pertaining to security matters comes to its attention at a later time). If a pre-determined mobile agent itinerary is not supplied by the agent user, the MASH may interface with a “yellow-pages” service to return a list of candidate non-overloaded-processing mobile agent platforms for the agent to perform some work on. The details of this process, including querying syntax and algorithm for determining which agent platforms have appropriate-related services is out of the scope of this thesis. However, from a security perspective, in Chapter 9 we present a preliminary reputation model for the MASHIn which can assist in the selection of agent platforms with reputable history of safe and reliable execution of agents.

**Steps 9-14:** The identity-anonimised agent travels (securely via asymmetric cryptographic messaging) around various agent platforms performing work, and its itinerary can also be anonimised via intermediate checking host routing (as this excludes prior itinerary hops from the knowledge available to subsequent agent platforms). The utilisation of checking hosts was first mentioned in Section 5.3.5 and is elaborated on in some depth in the next two chapters, especially Chapter 8.

**Step 15:** The completed agent is securely transported back from the MASH to its agent user (if desired, which we presume would be the rule not the exceptional case).

Mobile agent platforms will never see a mobile agent’s long-lived private key when mobile agents visit. Agent platforms will, if the mobile agent requires a temporary secret key, only be able to see a random short-lived (one hop, or series of trusted platform hops, useful) secret session key. In other words, if there is employment of cryptographic primitives requiring a secret key when a mobile agent is on an agent platform (e.g. for putting a time-limited hold on a shopping item) then the agent could use an anonymous session key generated by the MASH (or checking host) before the mobile agent is routed to its next untrusted agent platform.

Furthermore, as explained in the details for step 8 of Figure 6.5, the agent platform should not be able to identify the mobile agent user (by examining the mobile agent code or its packaging) because the mobile agent’s public keying
material including the corresponding agent user’s public key certificate has been stripped from the mobile agent packaging as the mobile agent has already been identified as an authenticated mobile agent entity at the MASH (or checking host) before routing it to the next agent platform. Instead, the mobile agent has a MASH-delegated proxy certificate attached to its packaging so that MASHIn mobile agent platforms can authenticate the agent. The agent user anonimisation process is the responsibility of the MASH Anonimisation Module which interacts with the MASH Authentication Module.

6.5 MASHIn Agent Location and Itinerary Event Monitoring Processes

In the MASHIn, a mobile agent’s itinerary position and timely mission completion can be tracked in a safe manner. This process is made possible by the fact that a MASH is responsible for mission protection of an agent. Before an agent’s location can be monitored by the dedicated TTP MASH, the executing agent instance must be uniquely identifiable in the MASHIn. This can be achieved by a one-way, collision-resistant, unkeyed cryptographic hash function [128] processed by the MASH before it forwards on the agent to do work remotely for the first time. For example, one possibility for the data input into this secure one-way is the concatenation of the MASH’s Distinguished Name, today’s date, and a sequential agent protection number\(^\text{12}\). Note, that for the purposes of ensuring agent (user) identity anonimisation, the input to this one-way hash includes no data relating to the identity of the agent user principal. Thus, even though the hash function is publicly known, agent platforms can not ascertain the agent user’s identity from brute-forcing inputs into the hash function to match the resulted output.

In Section 6.3.3, when discussing agent attribute certificates (and their extensions), we discussed the possibility of encapsulating Time-To-Live (TTL) requirements for an agent’s execution. Given this encapsulated data, the MASHs and checking hosts can safely monitor that agents do not experience delays at any one agent platform in their itinerary; this is useful in re-accommodating for problems arising from either a genuinely busy agent platform, or an attacking agent platform which is deliberately denying service (or an attacking agent denying service

\(^{12}\)For example, the \(n^{th}\) agent this MASH has been given responsibility for mission protection today.
for that matter). This security feature is reliant on the following two MASHIn design assumptions:

**Assumption 1:** MASHIn MASHs and (MASH-delegated) checking hosts are non-malicious entities to the major stakeholders (i.e. agent users and agent platform owners), and have not been compromised. They are, in fact, genuine TTPs to both stakeholders and highly reliable.

**Assumption 2:** Building on Assumption 1, the system clocks of MASHIn MASHs and checking hosts are tightly-synchronised.

We will now provide a simple example, in discussing Figure 6.6, of this MASHIn TTL itinerary event monitoring capability. Once again, the pictured itinerary of three agent platforms to visit is simply a convenient number for graphical-size reasons; the agent location and itinerary management scheme is flexible so that it can handle an arbitrary number of agent platforms.

Referring to the example presented in Figure 6.6 - on receiving an agent, authenticating it, and verifying the attribute certificate/s encapsulating privileged roles, the MASH (after either being provided with or determining the three untrusted agent platform itinerary) will coordinate in alliance with its delegated checking hosts the following event itinerary management and TTL monitoring steps. Once again, please note that all messages are sent securely using asymmetric cryptographic key messaging to ensure message sender authentication, message confidentiality, and message integrity.

**Preliminary Task (NOT shown on diagram):** The MASH first selects appropriate checking hosts. This could be based on a number of factors including - but not limited to - close proximity to the pertinent agent platforms, checking host current processing loads and response times, and a checking host’s level of mobile agent application-independence (i.e. to observe maximum objectivity in the checking host’s non-repudiation/checking responsibilities). Detailed criteria

---

13 Varying TTL for agent processing at different agent platforms and varying contingency policies in the event of itinerary hold-ups could also be supported.

14 Some agent platforms may be trusted by the agent user; if consecutive agent platforms in an agent’s itinerary are trusted by the agent user the insertion of a checking host may be deemed unnecessary. We will discuss this possibility in Chapter 8.
a MASH may employ in this checking host selection process, and algorithms for monitoring checking host and agent platform processing loads/response times are out of the scope of this thesis.

**Step 1:** Given the agent’s attribute certificate indicates that the first agent platform hop/execution has a TTL of 300 seconds (i.e. 5 minutes) and all subsequent hops/executions should have this same TTL, the MASH forwards to the first delegated MASHIn checking host the time of agent package dispatch\(^{15}\) to the first agent platform. Thus, this first chosen MASHIn checking host has been given responsibility for monitoring that no unreasonable delay in processing occurs at the first agent platform.

**Step 2:** The MASH forwards on the agent package to the first agent platform for execution, and indicates in this message that the agent’s next intended destination on execution completion there is (the IP address of) the first checking host.

**Step 3:** The agent platform finishes execution of the agent within the 5 minute TTL limit, sends on the agent package and it reaches the checking host within this TTL time limit.

**Step 4:** The checking host forwards on to the next delegated MASHIn checking host the time of agent package dispatch\(^ {16}\) to the next agent platform. Thus, this next checking host has responsibility for monitoring that no unreasonable delay in processing occurs on the second agent platform.

**Step 5:** The checking host forwards on the agent package to the second agent platform for execution, and indicates in this message that the agent’s next intended destination on execution completion there is (the IP address of) the second checking host.

**Step 6:** The second checking host, realising it has not received a message about the agent in a timely manner from the agent platform that was supposed to have executed and processed the agent (i.e. 5 minutes from its posting there), sends\(^ {17}\) the (copy of the agent received from the previous checking host) agent package to the next delegated MASHIn checking host (in this case the MASH itself will be acting as the checking host)\(^ {18}\).

\(^{15}\)And, a copy of the agent package in case something goes amiss in not being processed by the agent platform in a timely manner.

\(^{16}\)Once more, a copy of the agent package is also sent in case something goes amiss in the agent not being processed in a timely manner by the next agent platform.

\(^{17}\)This is by default, though some other action could be dictated by policy.

\(^{18}\)The MASH can, consulting policy, decide whether to eventually retry timely execution of
Figure 6.6: MASHIn agent location tracking and TTL processing example scenario.
Step 7: The checking host forwards on the agent package\textsuperscript{19} to the third agent platform for execution (since the second agent platform did not process the agent in a timely manner), and indicates in this message that the agent’s next intended destination on execution completion there is (the IP address of) the next checking host (which happens to be the MASH delegated mission protection for the agent).

Step 8: The agent platform finishes execution of the agent within the 5 minute TTL limit, sends on the agent package and it reaches the checking host (the MASH in this instance) within this TTL time limit.

Final Task (NOT shown on diagram): The MASH, on policy delegated from the agent user, can retry sending the agent to the original 2nd-in-line agent platform (e.g. if it is deemed mission critical) or perhaps be supplied with exception-handler code for late completing platforms (i.e. the 2nd checking host may have, since the agent reached the MASH, received the original agent which has completed execution on the 2nd agent platform)\textsuperscript{20}. In any case, once the MASH observes that it has appropriately handled the agent as best it can and/or the agent’s mission is complete, it will securely post the agent or its results back to the agent user - once again, as directed by policy delegated from the agent user.

The processing outlined above not only ensures that agents are processed in a timely manner, but facilitates that an agent’s location (and thus domain of accountable behaviour) can be non-refutably monitored by the TTP MASH which has been delegated agent mission protection (and utilising the assistance of checking hosts it selects before agent dispatching). A protocol which supports non-repudiation of mobile agent execution, designed primarily for ensuring agent execution integrity, will be detailed in Chapter 8. The MASHIn agent execution integrity protocol can work in either one of two modes for catching discrepancies in agent execution input claims made by the stakeholders. One mode is ‘real-mission-time’ whereby checking hosts immediately detect breaches of claimed inputs after an agent’s execution on an agent platform. The other mode is ‘directly-post-mission’ whereby the MASH can check for discrepancies of

\textsuperscript{19}If this has not been supplied by the previous checking host, it can be requested from that source.

\textsuperscript{20}Appropriate callback class logic, discussed in Chapter 7, could stipulate if executing the agent at agent platform 3 should proceed only if the agent has been executed at agent platform 2 first (i.e. order of agent execution on agent platforms may be significant in some contexts).
Another MASHIn itinerary design principle\textsuperscript{21} that facilitates safe management of remotely deployed mobile agents in the MASHIn is that dynamic changes to a mobile agent’s itinerary should only be permitted at the MASH which has been delegated agent mission protection or a supporting checking host. We will come back to the possibility of this trusted dynamic change to a mobile agent’s itinerary in Chapter 8.

6.6 Summary

In this chapter we have provided a solid overview and discussion of our MASHIn conceptual proposal for bringing the two major mobile agent stakeholders (i.e. agent users and agent platform owners) into an environment more conducive to interaction. The MASHIn is a novel large-scale, generic application, and generic agent system security strategy with an infrastructure-oriented perspective for bettering the prospects of Internet mobile agents. We have, in the course of this chapter, elaborated on some of the security services afforded by the MASHIn approach. We focused on the MASHIn TTPs - in particular the strategic responsibilities they contribute to attaining this overall MASHIn goal.

Section 6.1 provided details of the MASHIn objectives, the basic rationale for conceptualising such a high-level approach taking a broad-minded account of the interests of both major stakeholders, the major security services and their layered dependencies in the MASHIn, and some deployment considerations which need to be taken into account when considering the design and implementation of such a proposal. In Section 6.2 we detailed the necessary registration procedures both major stakeholders must undertake as a pre-requisite for participation in the MASHIn. The roles of TTP MASHIn-accredited registration and certificate authorities were discussed here. The role of TTP MASHIn-accredited attribute authorities in delivering a scalable and flexible privilege management model for the MASHIn was discussed in Section 6.3. The process of facilitating scalable MASHIn authentication and anonimisation services was discussed in Section 6.4, accommodating significant reductions in key management concerns for both major stakeholders. Finally, in Section 6.5 we discussed the inherent secure agent

\textsuperscript{21}This is the default recommended policy, though it could be overridden if an agent user desires.
location and itinerary management services afforded by the MASHIn design which boost the robustness of an agent’s mission completion chances and also have utility in lowering the load of agent platforms who are either genuinely busy or perhaps under attack.

The remaining thesis chapters provide more details on our MASHIn conceptual offering. In the next chapter (i.e. Chapter 7) we discuss a novel mobile agent code and data privacy mechanism we have devised for the MASHIn. Chapter 8 provides details of an adapted mobile agent execution integrity protocol. Appropriately utilising this mechanism in the context of the MASHIn overcomes the two major weaknesses of the scheme identified by its designer, and offers a means for tracking malevolent inputs by either stakeholder in a mobile agent’s work on remote agent platforms, both in real-time or post-mission - in either mode providing a significant deterrent against malevolent stakeholder behaviour. The deterrence theme is elaborated on in Chapter 9 when we present the basis of a robust reputation model for the MASHIn which has flexibility for considering stakeholder input criteria in calculating personalised, high-quality reputation accounts of potential stakeholders and at the same time provides a method for bringing previously unknown (and thus implicitly mistrustful) stakeholders together for the initiation of useful work. In Chapter 10 we consider precautionary methods for protecting MASHs, the central pivotal TTP in the MASHIn from which a lot of security functionality is either directly encapsulated or delegated.
Chapter 7

Protecting Mobile Agent Data and Code Privacy

If the facts don’t fit the theory, change the facts.

Albert Einstein

In the previous chapter we provided a solid discussion of the rationale for, and entities constituting, the MASHIn concept - our secure strategy for bettering the prospects of Internet mobile agents by bringing the two major stakeholders into an environment more conducive to interaction.

In this chapter we present our novel and effective application-independent mobile agent data and code privacy strategy that we have devised and titled Secure Callback Classes for Mobile Agent Privacy (SCC4MAP). SCC4MAP is a mechanism for achieving high quality mobile agent privacy and is readily practicable within the MASHIn framework. In Section 7.1 we commence the chapter by discussing the broad objectives of mobile agent data and code privacy that we are interested. From an analysis of the pertinent body of research literature, in Section 7.2 we identify the pros and cons of existing mobile agent privacy strategies. Then in Section 7.3 we present the rationale for our SCC4MAP mobile agent privacy mechanism and detail a proof-of-concept SCC4MAP prototype we built and successfully tested. In Section 7.4 we compare and contrast the existing mobile agent privacy approaches with our new SCC4MAP strategy. Finally, in
Section 7.5 we conclude the chapter with an appraisal of our novel contribution and the applicability of the SCC4MAP mobile agent privacy mechanism to the MASHIn proposal.

This chapter includes and builds on material found in peer-reviewed accepted papers [i]-[iii] listed on page xxi.

7.1 The Objectives of Mobile Agent Data And Code Privacy

It is difficult to clearly explain what the objectives of ‘data’ and ‘code’ mobile agent privacy are without discussing an application, so we will consider an application example. We discuss a case scenario involving a purchase shopping mobile agent to facilitate understanding of the data and code privacy objectives, though we stress again that the MASHIn security proposal is application-independent and can be used as the basis for securely supporting arbitrary mobile agent application types.

A “purchase shopping mobile agent” is an extension of the “window shopping mobile agent” discussed in Section 5.1.4 in that not only does such a mobile agent travel around various agent platforms searching for the best price for an item (or bundle of items), it purchases the best-priced item. An extension to this agent mission task, facilitated in our SCC4MAP prototype discussed in Section 7.3, is that the purchase mobile agent should initiate the purchase of a ‘bargain-priced’ item immediately it is quoted an offer at least as good as a pre-determined bargain price.

We now pictorially model the possible objectives of a traditional (i.e. non-MASHIn) purchase shopping mobile agent, representing two different scenarios. In Figure 7.1, depicting the first scenario, a purchase mobile agent is sent to a number of agent platforms to negotiate quotations for a “Gizmo” (this could be the unique identifier for a single item, or a bundle of items). The agent knows that it should purchase a ‘bargain’ Gizmo immediately such a price (e.g. $1250 in the scenario pictured) is quoted. The agent has an itinerary of three agent platforms; on travelling to and negotiating with each in turn it realises that no agent platform quotes a bargain price. At the end of its travel to these three agent platforms, the agent proceeds to buy the best-priced Gizmo on behalf of its user. In this scenario, the best price quoted was from agent platform 1.
7.1. The Objectives of Mobile Agent Data And Code Privacy

Figure 7.1: Purchase agent buys the best-priced Gizmo.

Figure 7.2: Purchase agents impulsively buys a bargain-priced Gizmo.
The second scenario, in Figure 7.2, depicts that a purchase agent once again has an itinerary of three agent platforms to visit and negotiate with for a best or bargain priced Gizmo, however when it reaches the second agent platform, the agent platform quotes a price for the Gizmo at least as good as (in fact better than) the agent’s bargain price. The agent, realising that this is a bargain, initiates a purchase operation buying the Gizmo on behalf of its agent user.

There is an additional criterion possible for the purchase agent (not pictured). The agent user may wish for the agent to only buy a best-priced Gizmo if it is at least as inexpensive as a maximum price the agent user is willing to pay for the Gizmo.

A number of security issues can be pondered with respect to the purchase shopping agent, for example what mobile agent data fields (immutable and mutable - i.e. agent-platforms-to-visit/search-item/bargain-price and gathered-prices respectively) is visible to the agent platforms, and how and from where does the purchase agent initiate buying the best-priced Gizmo. We repeat that the MASHIn is mobile agent application-independent. The MASHIn provides purchase shopping mobile agents and arbitrary other types of application agents with a number of safeguards. Later, in Section 7.3, when we revisit the purchase shopping mobile agent application example, we will address these and other issues, specifically in the light of accommodating application-independent MASHIn safeguards.

What is pertinent, from a mobile agent data and code privacy perspective, to highlight now from the purchase shopping mobile agent application example is that, if the agent user desires, the following details should be kept secret from agent platforms:

- The price thresholds (i.e. bargain and maximum purchase prices) are examples of mobile agent \textit{data privacy}\footnote{The agent platform input quotation prices could also be thought of as requiring mobile agent data privacy, but these are agent-via-agent-platform acquired data items (not sensitive mobile agent data \textit{per se}). We discuss, in Section 7.3.5, how these agent platform input values could be protected for privacy in the MASHIn context.}, and

- The fact that it is a purchase agent (i.e. willing to buy not just window shop) and how it goes about buying the best-priced or bargain-priced Gizmo is an example of mobile agent \textit{code privacy}.

In the next section we analyse existing approaches located in the body of
mobile agent security research literature for achieving mobile agent data and code privacy.

7.2 Existing Strategies to Achieving Mobile Agent Privacy

Achieving mobile agent data and code privacy (i.e. from a prying attacking “agent platform’s eyes”) is not possible in the general case because the agent platform has complete access to the mobile agent’s code. We wish to accentuate the following important point - different agent systems and different agent applications store agent data in a variety of ways, including - but not limited to - as instance variables of the mobile agent object or within an agent encapsulating package such as a JAR-file. Sometimes the delimitation between an agent’s ‘data’ and ‘code’ can therefore be misunderstood. The MASHIn, because it is an agent-system-independent and agent-application-independent security proposal, is generic driven and capable of supporting a variety of underlying agent data storage structures. Most importantly, regardless of one’s opinion on what constitutes ‘agent data’ and ‘agent code’ per se, the MASHIn mobile agent privacy mechanism (discussed in Section 7.3) is capable of achieving agent data and code privacy across the board.

A number of mechanisms for achieving mobile agent privacy have been proposed which either broadly strategise to:

1. Make the adversary’s (i.e. an attacking agent platform’s) agenda more difficult, or

2. Solve the general case agent platform adversary problem. As we discuss in Section 7.3, our SCC4MAP mobile agent privacy mechanism would be categorised in this second, stronger set of strategies.

In Section 7.2.1 we present an appraisal of the existing approaches that we feel are best categorised within the set of techniques employing the first broad strategy, and in Section 7.2.2 we critique the existing approaches that we feel are best categorised within the set of techniques employing the second broad strategy. In Section 7.2.3 we present some summary comments on the existing strategies for achieving mobile agent privacy.
7.2.1 Approaches Which Make the Adversary’s Goal More Difficult

The following two strategies are ‘weak’ mobile agent privacy mechanisms because given feasible resources (time, knowledge, and/or hardware) - significantly less resources needed than brute-forcing your average asymmetric private key - and active monitoring and manipulation, the techniques’ modus operandi and output can be reverse-engineered or analysed (and beaten) by attacking agent platforms.

7.2.1.1 Obfuscation Strategies

Obfuscation, or as Hohl terms his obfuscating process “Code Mess-Up” [83], is a non-cryptographic mobile agent protection scheme that can be employed for mobile agent privacy which is nevertheless built-up like any cryptographic mechanism: input (i.e. agent code and data) is transformed into an unreadable form\(^2\) by a mechanism that should not be inverted easily with current knowledge. However, obfuscation techniques do not offer long-term protection in comparison to cryptographic techniques. Despite various forms of obfuscation (including Layout, Data, Control and Preventive Obfuscation [230, 77, 39, 14] techniques) and a number of attempts to apply obfuscation algorithms to mobile agents [83, 84, 88, 48], the conclusions reached are not promising from a practical mobile agent privacy perspective.

D’Anna et al. [48] stated that even though obfuscation could delay an agent platform’s perspective on the internals of an agent, reverse engineering attacks could not be prevented and with enough computational resources, such as enough time, the code and data can always be de-obfuscated. Their team at Network Associates Laboratories studied state-of-the-art mechanisms in obfuscation and built a number of varying strength obfuscation tools and techniques. On building tools to de-obfuscate code obfuscated by their most powerful methods, they observed that complete de-obfuscation was disappointingly easy to achieve.

Barak et al. [14] studied the theoretical limits of obfuscation techniques (i.e. via underlying mathematical principles) and showed that, in general, achieving completely secure obfuscation in an efficient practical manner is impossible. Simplistically stated, their major conclusion was that obfuscated programs tell the form of their original program in one of many possible ways.

\(^2\)While preserving its original intended program functionality.
Two of the interesting conclusions that D’Anna et al. make from their comprehensive obfuscation review and investigation that we think are pertinent in this discussion are [48]:

1. There is no reason at all to depend on obfuscation for security. This is not to say that obfuscation should not be used, but high-value secrets must not be entrusted to it.

2. Because obfuscation is rather difficult to implement, untrustworthy, and has a history of being broken, we suggest that obfuscation not even be considered until all other possible approaches have been ruled out.

Significant main challenges remain before obfuscation for mobile agent privacy can be considered a reliable security mechanism including formulating a usual time-period protection guarantee when applying general obfuscation techniques, how to make obfuscation processes transparent and usable in mobile agent systems, and researchers need new insights that go beyond currently known techniques in order to come up with an effective obfuscator [84, 96, 14, 48].

### 7.2.1.2 Environmental Key Generation Towards Clueless Agents

Riordan and Schneier [162] describe a proposal whereby an agent takes some stealthy action on some environmental condition becoming true. The environmental condition, which is hidden through a one-way hash or public key encryption, is a trigger for a key to be generated and is used to unlock some executable code and/or data cryptographically. Without the environmental condition (which is not immediately clear from the agent’s perspective) being met, the agent is unable to decrypt its own message (encapsulating the sensitive code and/or data) [96, 5]. In this sense, the agent is termed “clueless” of its hidden true mission by the scheme’s proposers.

Whilst this technique ensures that an agent platform or an observer of the agent cannot immediately uncover the triggering message or response action by directly reading the agent’s code, the environmental key generation mechanism is not a reliable means for achieving mobile agent privacy. This is because the agent platform could alter the interpreted sequence of agent operations to instead just print out the executable code and/or data upon the event of the environmental condition becoming true triggering the generation of the activation unlocking key, instead of proceeding to execute the agent. Moreover, most mobile agent
Chapter 7. Protecting Mobile Agent Data and Code Privacy

environments - including the Java Virtual Machine\(^3\) - do not permit the dynamic generation of code at runtime in following procedures for the interpretation of untrusted code (e.g. the code generated may be a virus and cannot be checked for safety using the normal precautionary methods) [162, 96, 5].

7.2.2 Approaches Which Solve the General Case Adversary Complexity

The following strategies are generally sounder, from a pure security perspective, than the ‘weak’ approaches discussed in Section 7.2.1 for achieving mobile agent privacy because they are not susceptible to agent platform analysis. This is due to the fact that an agent platform must rely on breaking some sound cryptographic process or collaborate with agent platforms trusted by the agent user.

7.2.2.1 Computing with Encrypted Functions

E-commerce and mobile agent security research has sought a method to compute encrypted functions on the agent platform for many years. Several researchers have sought methods to encrypt a function, send it to a remote location, and execute it such that the execution environment could obtain the result but not know the function [48]. The most promising results, from the perspective of autonomous mobile agents, were those given by Sander and Tschudin [175, 174].

Sander and Tschudin [175, 174] created quite a boost of optimism in late 1997/early 1998 for the mobile agent community (and wider mobile code community) when they published a technique showing preliminary evidence of safely computing agent cryptographic primitives, such as a digital signature, even though the code is executed on untrusted agent platforms\(^4\) [174]. The strategy is to have an agent platform execute a program embodying an enciphered function without being able to discern the original function; the strategy requires differentiation between a function and a program that implements that function [96].

An extension of the encrypted function concept was that of computing with encrypted data. To prevent the agent platform from spying out private data, the encrypted functions must be homomorphic in relation to the original functions. The homomorphic property allows an encrypted function to operate on encrypted

---

\(^3\)The foundation on which most mobile agent systems are built.

\(^4\)And, importantly, without interaction with the agent’s home platform.
data and produce a result that can be translated to an equivalent result if a non-
encrypted function and data are used [174, 40].

A mechanism like the computing with encrypted functions process would be
ideal for mobile agent privacy if it were possible to find appropriate encryption
schemes that would transform an arbitrary function \( f \) (i.e. any mobile agent
program and encapsulated data). At the moment, however, appropriate classes
of functions which can be encrypted as proposed by this scheme are limited
only to polynomial and rational functions - a very small subset of all possible
functions [174, 1, 5, 96].

### 7.2.2.2 Secure Computation Service

Algesheimer et al. [6] argue that the approach based on computing with encrypted
functions [174] just discussed only works if nobody but the user learns the result of
the computation. As the agent platform can observe the output of the encrypted
function, it can simply re-run the agent’s encrypted code again with different
inputs. The agent platform would then be able to learn the functionality of
the encrypted code by simply trying many different inputs and looking at the
outputs. Moreover, the agent platform could choose and send back the output
it views as most appropriate. If the agent platform is to receive some output
of the computation then providing execution privacy requires minimal trust in
a third party (or in a tamper-resistant module\(^5\) at the agent platform). They
present a solution in which a generic independent online entity they refer to as
the *secure computation service*, performs some operations on behalf of the mobile
agent but does not learn anything about the encrypted computation except its
output size. This universal solution is applied to agents that visit more than one
agent platform before they return home to the agent user; the secure computation
service is invoked when the agent travels from one agent platform to another agent
platform [38].

The secure computation service approach has several benefits for attaining
mobile agent privacy including it can serve (in theory) arbitrary applications,
agent platforms gain nothing from attempting to re-run agents with different
inputs, secure computation servers may be set up and operated by independent
entities, and the secure computation servers can be built using software and

---

\(^5\)We have discussed in Section 5.1.3 why we do not consider tamper-resistant hardware
modules at agent platforms as a widespread solution to mobile agent security issues.
commodity hardware (instead of trusted modules/dedicated hardware) resulting in reduced start-up and subscription costs [6].

However, there are at least a few disadvantages of the scheme including:

- The secure computation servers may become overloaded or be targeted via denial-of-service attacks (and if they do crash or become unusable, another secure computation server cannot be incorporated due to the cryptographic key management employed in the scheme’s protocol) [48],

- All parties - including all agent platforms in an agent’s itinerary - must follow the schemes’ cryptographic and message protocol steps correctly for the agent’s mission to have a chance of success [6],

- The associated cryptographic costs of the privacy scheme are prohibitive for large functions of mobile agents [6, 48], and

- Any agent that requires frequent interaction with an agent platform (e.g. for database searches) would be completely unsuitable, as the agent would have to contact a trusted server after each interactive round of communication [48].

Yee [237] concludes that mobile agent privacy approaches such as the computing with encrypted functions and secure computation service are extremely expensive in terms of cryptographic overheads, have very restricted domains of application viability, and are not ever likely to be practical.

7.2.2.3 Code Splitting Among Cooperating Agents

Roth concludes that protocols based on cooperating agents have merit since they are less susceptible to attacks by coalitions of agent platforms than single agents [164]. By dividing up the operations (and possibly data) of an application agent between two (or more) cooperating agents, certain malicious behaviour of an agent platform can be detected or perhaps prevented [164, 237, 147, 126, 96].

There are quite a few drawbacks to these types of schemes however:

- Setting up an authenticated channel for the cooperating agents to securely communicate can be expensive. Moreover, if an agent platform refuses to provide an authenticated communication channel, the cooperating agent protocols break down [96, 5].
7.2. Existing Strategies to Achieving Mobile Agent Privacy

- If an agent is killed, it could be difficult for the peer to determine which agent platform was responsible [96, 5].

- It is necessary to determine two disjoint agent platform itineraries, with at least one agent from either itinerary being located on a trusted platform at any one time. This may not always be feasible, which could mean one of the agents remains stationary at a trusted agent platform - in the worse case the home agent platform - thus breaking the definition of autonomous mobile agents with agent user (and home platform) capable of disconnection from the network [5, 164].

The Remote Digital Signing with Mobile Agents group threshold scheme was discussed in Section 4.2.4.5, and the Self Protecting Mobile Agents proposal was discussed in Section 4.2.4.6; both schemes incorporate multi-agent splitting of sensitive code/data parts for proposing to support digital signature\(^6\) generation and generic mobile agent functions respectively.

7.2.2.4 Supervisor-Worker Framework

The Supervisor-Worker Framework [61] approach could, in one respect, be thought of as fitting within the previous category of code splitting among cooperating agents group of mobile agent privacy mechanisms. Code and data in this approach is dispersed among cooperating agents, but the cooperating model is different with a supervisor agent coordinating all of his worker agents. The supervisor agent, while capable of mobility, must be located at a trusted platform at all times. The supervisor retains the bulk (if not all) of the sensitive code and data (of the agents’ collective mission), while it dispatches worker agents to less trusted agent platforms for gathering or depositing information as needed.

There are some positive aspects to this approach including the fact that security is built in at agent design-time with care given to the agents’ separation of concerns, the supervisor can resend or determine alternative measures if one or more of his workers are not responsive, and because the supervisor is located at a trusted location the goal of maintaining agent code and data privacy is achievable.

However, there are also a few drawbacks to the Supervisor-Worker Framework including:

\(^6\)With shares of the private key distributed.
• If there are no trusted agent platforms available, the supervisor agent must reside and work from its home (i.e. agent user’s) platform - but, the home platform may have disconnected from the network. The framework then falls apart. Even if the home platform is still connected to the network, because autonomous work is not done solely at remote agent platforms the general understanding of autonomous mobile agent work is violated.

• There is significant time involved in preparation, for example the high-level non-trivial nature of the decision making involved in defining constraints for the supervisor mobile agent, and/or in real-time, to ensure the safety of his workers (and ultimately the mobile agent’s mission). This places a significant extra, quite likely, unwanted burden on the mobile agent’s designer and/or user.

• There is added network traffic and supervisor coordination required springing from each worker agent dispatched.

7.2.3 Summary of Existing Mobile Agent Privacy Strategies
In Section 7.2.1 we discussed two ‘weak’ techniques for achieving data and code privacy in mobile agents. They were dubbed ‘weak’ because their modus operandi is not based on secure principles, but rather clever ‘tricks’ which given enough time or diligent real-time analysis can be subverted by an attacking agent platform. In Section 7.2.2 we discussed four techniques for achieving mobile agent privacy which rationale their modus operandi on sounder security principles, specifically either employing cryptographic mechanisms or utilising trusted platforms in their scheme. Nevertheless, we identified and discussed a number of advantages and weaknesses in each of the six schemes.

Table 7.1 presents a summary analysis of the existing approaches, in particular our appraisal of the practicality, effectiveness, and efficiency of the schemes with respect to achieving mobile agent (data and code) privacy for arbitrary deployed mobile agent applications.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Obfuscation</td>
<td>Not easy to target specific code and data for protection</td>
<td>Low-Medium (deploying and utilising non-cracked obfuscation tools is not a transparently simple exercise for the average end-user)</td>
<td>Low-Medium</td>
<td>Poor set-up at home platform, negligible interpretation penalty at agent platforms</td>
</tr>
<tr>
<td>Environ. Key</td>
<td>All or nothing</td>
<td>Low (most agent systems - i.e. built on JVMs - will not allow dynamic code generation)</td>
<td>Low</td>
<td>Depends on the form, and timeliness, of the trigger</td>
</tr>
<tr>
<td>Generation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comp. w/ Enc. Funct.</td>
<td>Ingenious but limited in application</td>
<td>Low (very small subset of functions can be viably encrypted and executed to meet the scheme’s aims)</td>
<td>Medium-High</td>
<td>Moderate, but highly crypto-intensive</td>
</tr>
<tr>
<td>Secure Computation</td>
<td>Good for small functions, inappropriate for large functions</td>
<td>Low-Medium (even though TTP servers could perform a service for small functions, the protocol design means agent mission completion is not robust if something goes wrong with any part of the protocol e.g. any stakeholder denying service)</td>
<td>High (for small functions)</td>
<td>Double the general case messages on migration, and highly crypto-intensive</td>
</tr>
<tr>
<td>Service</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cooperating</td>
<td>Unclear how it can be applied to arbitrary agents</td>
<td>Medium (deciding safe mutual routes would not always be a simple task, and the protocol is not robust should an agent be killed or denied an authenticated channel)</td>
<td>Medium-High</td>
<td>Poor (more agents and more messages)</td>
</tr>
<tr>
<td>Agents</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supervisor-Worker</td>
<td>Reasonably simple to design and use</td>
<td>Medium (locating a trusted platform for the supervisor to work from may be difficult, and if many supervisors are coordinating from the one agent platform this would be resource intensive)</td>
<td>High</td>
<td>Relatively good except that the platform hosting the supervisor could become bottlenecked</td>
</tr>
<tr>
<td>Framework</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.1: Summary analysis of existing approaches supporting mobile agent privacy.
In the next Section (i.e. Section 7.3) we provide a detailed discussion of our novel approach to mobile agent privacy which, coincidentally, fits in well within our broader MASHIn framework. We come back to Table 7.1 in Section 7.4 when we compare the pros and cons of our proposed new mobile agent privacy strategy against the existing strategies from the body of research literature reviewed in this section for achieving mobile agent privacy.

7.3 Secure Callback Classes for Mobile Agent Privacy

We have produced, and tested the viability of, a novel strategy for achieving mobile agent privacy that we titled Secure Callback Classes for Mobile Agent Privacy (SCC4MAP). We present the rationale for our novel SCC4MAP mechanism in Section 7.3.1. Section 7.3.2 provides an overview of our scheme, with a proof-of-concept implementation discussed in Section 7.3.3. Sections 7.3.4 and 7.3.5 discuss extensions to the proof-of-concept, specifically for demonstrating the additional incorporation of checking hosts for handling of sensitive mobile agent operations and data\(^7\) and facilitating mobile agent platform privacy of input values.

7.3.1 Rationale for Strategy

In designing the SCC4MAP mechanism we took on board the weaknesses of existing mobile agent privacy mechanisms as summarised in Section 7.2.3. Moreover, the SCC4MAP mechanism was motivated by the following MASHIn objective listed in Section 6.1.1: *The sensitive operations of a mobile agent should be kept totally inaccessible to untrusted mobile agent platforms. Agent platforms should not be able to subvert agent code to utilise sensitive operations inappropriately, and they should learn less of the mobile agent’s susceptibilities/thresholds.*

We achieved not allowing agent platforms to subvert agent code to utilise sensitive operations by keeping the sensitive operations visible only to trusted MASHs and checking hosts as the agent goes about its mission. Likewise, the agent’s susceptibilities/thresholds are made known only to these two MASHIn

\(^7\) The implemented SCC4MAP proof-of-concept only incorporated the notion of a TTP MASH, as this was sufficient to test the viability of the SCC4MAP mechanism.
TTP entities, not agent platforms.

### 7.3.2 Strategy Overview

A fundamental mechanism available to mobile agents through TTP MASHs and their delegated TTP checking hosts are secure event callback methods. Referring back to the mobile agent life-cycle discussed in Section 2.3, we are proposing events in and around the *migrating* state of a mobile agent. These events correlate to certain points in the mobile agent’s itinerary - specifically either one of: an itinerary milestone (after each hop to a target agent platform, after a specific hop to a specific target agent platform, or after all hops have been completed); or before or after a mobile agent hops to an agent platform that is deemed outright trusted/untrusted or having a finer-grained trust weighting to the mobile agent user.

An extensible base set of interface class implementation methods is sent with the agent on the mobile agent user’s mission protection request to a MASH. The base set of methods in this interface (and thus also the concrete class as well) include:

- **afterEveryHost()**: Triggered on the MASH\(^8\) or a checking host after every hop to an agent platform.

- **afterNthHost(int[] nthHostList)**: Triggered on the MASH or a checking host after a hop to a specific agent platform in a pre-determined itinerary list. The list of pertinent agent platforms to invoke this method on after visitation is passed in as an integer array.

- **beforeNonTrustedHost()**: Triggered on the MASH or a checking host before the agent hops to an agent platform specified as non-trusted\(^9\).

- **afterNonTrustedHost()**: Triggered on the MASH or a checking host after the agent is finished working at, and hops from, an agent platform specified as non-trusted.

---

\(^8\)Whilst there are arbitrarily many MASHs in the MASHIn, when we refer to the “the MASH”, we are referring the the particular MASH which has been given the responsibility for monitoring this agent’s safe mission completion.

\(^9\)Trust here is in fact handled as a boolean, either trusted or non-trusted but the before/afterNonTrusted(int trustLevel) methods support finer-grained (trust level) ⇒ (code sensitive visibility clearance) handling.
• **afterMission()**: Triggered on the MASH after the agent’s mission is finished and at the MASH for the last time before being scheduled for shipping back (or alternate-specified action) to the mobile agent user.

• **beforeTrusted(int trustLevel)**: Triggered on the MASH or a checking host before the agent hops to an agent platform specified as having at least as great a trust level as the [0-100] (i.e. percentage) integer passed into the method.

• **afterTrusted(int trustLevel)**: Triggered on the MASH or a checking host after the agent is finished at, and hops from, an agent platform specified as having at least as great a trust level as the [0-100] (i.e. percentage) integer passed into the method\(^\text{10}\).

Note the above method names were chosen arbitrarily by us. There is no reason an agent user could not use arbitrary other naming, so long as they are consistent in their handling of events via the pertinent callback class methods, and still benefit from our SCC4MAP mobile agent privacy offering.

The supplied mobile agent logic, privy only to the TTP MASH and its TTP delegate checking hosts, can then be used to support the autonomy and security goals of the mobile agent in our proposed infrastructure. The SCC4MAP modus operandi to achieving mobile agent privacy involves denying accessibility, and thus visibility, to the sensitive mobile agent data and operations (i.e. ensuring mobile agent data and code privacy respectively). The mobile agent’s autonomy is ensured; no interaction post-deployment is necessary with the agent user, due to the incorporated TTP MASH and checking hosts, for the agent to fulfil its mission safely (and dynamic changes to the agent’s itinerary can be determined by the agent on the TTP MASH or a checking host). Figure 7.3 provides a pictorial representation of our SCC4MAP scheme.

For an implementation in Java, a mobile agent user will represent an agent in a JAR file with three constituent parts ready for a MASH:

• **Driver mobile agent classes**: These are the main invocation classes of the mobile agent application instance, privy to both MASHIn TTPs and agent platforms that the agent visits during its mission.

\(^{10}\text{In Chapter 9, we propose a novel mechanism for enabling a reliable means of calculating personalised trust levels which can be updated periodically or in real-time within the MASHIn framework.}\)
7.3. Secure Callback Classes for Mobile Agent Privacy

Figure 7.3: A representation of the flow of classes in our SCC4MAP scheme.
• **'Dummy' callback classes**: These sets of class files\(^{11}\), invoked by the agent’s driver classes, are transported to untrusted agent platforms, and should therefore contain non-sensitive data and operations only - perhaps acting as a means of distraction to (possibly) malicious-intending agent platforms.

• **'Real' callback classes**: These sets of class files\(^ {12}\), invoked by the agent’s driver classes, are only accessible to (and interpreted by) MASHIn TTP MASHs and checking hosts in the agent’s itinerary.

The MASH (and checking hosts) will then only forward on the agent driver classes and ‘dummy’ classes to untrusted agent platforms, and only forward on the ‘real’ callback classes to incorporated checking hosts.

Because a mobile agent needs access to all its specified classes with it at runtime and as it travels, a non-trivial mechanism for inserting and deleting a ‘real’ versus ‘dummy’ callback class was needed. However, the large majority of existing mobile agent systems (especially those built on JVMs) - quite rightly - do not allow for this dynamic changing of classes as an obvious precautionary measure in the protection of agent platforms.

We therefore decided to implement our own basic agent system as a proof-of-concept that we could achieve this MASH/Checking Host-privy only logic callback loading and unloading mechanism. The crux of the solution is writing a custom class loader. Our proof-of-concept was demonstrated using Java, so we extended Java’s `ClassLoader` class to look up a local repository of classes (which could be updated appropriately) on the TTP MASH. As noted earlier, the proof-of-concept did not incorporate the utilisation of checking hosts, so secure transport and loading of the ‘real’ callback class between a MASH and checking host was not implemented. This is, however, a trivial extension discussed in Section 7.3.4 to the non-trivial successful proof-of-concept base idea of the JVM secure callback classes (i.e. ‘real’ and ‘dummy’) loading/unloading process.

\(^{11}\) Usually incorporating only one ‘dummy’ callback class will suffice, though an arbitrary number can be supported depending on the level and form of abstraction required by the application.

\(^{12}\) Again, incorporating only one ‘real’ callback class will suffice, though the number must match the number of employed ‘dummy’ callback classes.
7.3.3 Proof-of-Concept Prototype Application

As alluded on in Section 7.3.1, the approach to achieving mobile agent data and code privacy we set to undertake, and achieved, involves - to the best of our knowledge - an original approach.

Due to the reasonable JVM safety provision which does not permit dynamic updated class definitions being loaded/unloaded at runtime, we wrote our own little agent system when implementing the SCC4MAP proof-of-concept. The whole proof-of-concept including our own agent system with custom class loader was achieved in only approximately 1500 lines of Java code\textsuperscript{13}, with the agent server representing the TTP MASH receiving from the client\textsuperscript{14} agent user a jar with the driver agent class, the ‘real’ callback class, and the ‘dummy’ callback class. Finally, another file was included in the jar to signal to our agent system which class was the driver class\textsuperscript{15}.

For the proof-of-concept, the agent driver class was called \texttt{PurchaseAgent} implementing Java’s \texttt{Runnable} (for commencing a separate thread of execution) and \texttt{Serializable} (for marshalling the agent and storing it in a format suitable for exchange over the network, and then to be deserialized and interpreted by the next agent platform in its itinerary).

The ‘real’ callback class was called “\texttt{CallbackClass-REAL.class}” and the ‘dummy’ callback class was called “\texttt{CallbackClass-DUMMY.class}”. When these two files were created, they were compiled from two distinct “\texttt{CallbackClass.java}” files and then subsequently renamed to the aforementioned two ‘real’ and ‘dummy’ signifying class name files for the TTP MASH (and checking hosts, if the proof-of-concept was extended to simulate their involvement). The TTP MASH is responsible for copying and renaming the appropriate \texttt{CallbackClass-(REAL/DUMMY)-.class} to \texttt{CallbackClass.class} at the needed times - specifically (for the ‘real’ callback class) before the agent is executed on the TTP MASH or a checking host, and (for the ‘dummy’ callback class) before the agent is shipped to a non-trusted agent platform. Since the main agent driver class only invokes an object of type \texttt{CallbackClass}, this correct renaming of ‘real’/‘dummy’ classes into

\textsuperscript{13}SCC4MAP prototype class descriptions and relationships are discussed in more detail in Appendix B.
\textsuperscript{14}As discussed in Chapter 10, for security precautionary reasons, in a live practical MASHIn environment client agent users would not be able to directly contact a MASH.
\textsuperscript{15}In a production agent system, indicating an agent’s driver class can be done with more purity and transparency via meta-data in a JAR-file’s manifest [181], however for the purposes of the proof-of-concept our additional label-driver file was satisfactory.
CallbackClass is a pre-requisite to prevent a halting runtime exception by the agent platform’s class loader.

The PurchaseAgent visits a number of remote agent platforms and has an integer variable indicating which agent platform it is on, incremented by 1 (in the agent’s run() method) each time the agent is invoked on an agent platform in its itinerary. The agent then queries the agent platform for an offer price (for an item, or ticket, or service, or bundle of any of those - for whatever it is shopping for in the hope to purchase). If the agent platform’s offer is better then best offer so far, a record of this (the new best offer and agent platform number) is updated in the purchase agent.

The (PurchaseAgent) driver class then instantiates a CallbackClass object via Java’s new keyword operator (for creating objects). The afterEveryHost() method of the CallbackClass is then invoked. Because the agent is on a untrusted agent platform, the agent platform only sees an empty method implementation in the ‘dummy’ callback class. However, when on the TTP (MASH or checking host), the ‘real’ callback class is invoked. In this ‘real’ implementation of afterEveryHost() is a test to see if the best price offer so far in the agent’s mission is equal or better (i.e. cheaper) than a pre-determined ‘bargain’ price (for which it only knows - stored only in the ‘real’ callback class). If this indeed the case, the ‘real’ callback class proceeds to initiate purchase of the bargain\textsuperscript{16} otherwise the agent proceeds to the next agent platform in its mission.

If at the end of the agent’s full itinerary, a ‘bargain’ price has not been reached, the afterMission() method of the ‘real’ CallbackClass is invoked on the TTP MASH. The implementation of afterMission(), privy only to the TTP MASH (and possibly incorporated checking hosts), checks to see if the best offered price thus far is equal or better (i.e. cheaper) than a pre-determined ‘acceptable’ price (for which it only knows). If an acceptable (but not bargain price) was offered, the ‘real’ callback class proceeds to purchase the acceptable-priced item, otherwise it does not buy anything because no acceptable price was found.

For example, a bargain price might be $1500, and an acceptable price might be $1700 (both of whose values are only known to the TTP MASH via the purchase agent’s ‘real’ CallbackClass). After every agent platform, if an offer was

\textsuperscript{16}In the proof-of-concept, this is an empty purchase event (though it could have been the TTP purchasing as a broker with some deposit provided in advance by the agent user; in a real-world application which is extra security conscious this may involve putting a ‘time-limited hold’ on the purchase hold until the agent user can complete the purchase directly.
made less (i.e. cheaper) than or equal to $1500, the agent would purchase the bargain immediately (and its mission is completed!), otherwise it would proceed to the next agent platform in its mission itinerary. If at the exhaustion of that mission’s itinerary no bargain was found but the best offer made was cheaper or equal to $1700, the agent would purchase that best (acceptable offer). However, if no acceptable offer was made by any agent platforms, then the agent does not make a purchase.

In the context of the above proof-of-concept application example, the pre-determined ‘bargain’ and ‘acceptable’ prices are our new example of mobile agent data privacy, only declared in the ‘real’ callback class, privy only to the TTP MASH (and possibly checking hosts). Similarly, the non-empty method implementations of afterEveryHost() and afterMission() are our new examples of mobile agent code privacy, only listed in the ‘real’ callback class.

The only potentially sensitive data in the purchase agent example that was privy also to untrusted agent platforms is the value of the best price offered thus far. However, this may be viewed as an appropriate bargaining tool visible to the ‘eyes’ of the agent platform - such that the challenge “Can you beat this price that I have been offered?” is subliminally posed to the visited agent platform. In any case, there is no reason why this best price variable cannot be stored on a TTP MASH and not in the driver purchase agent class, so this too can be kept privy only to the TTP MASH.

In summary, our proof-of-concept for demonstrating the feasibility of our novel approach to mobile agent privacy (i.e. SCC4MAP) was successful. In the proof-of-concept, we used a purchase agent application example; however our SCC4MAP approach is readily applicable to arbitrary application contexts and arbitrary agent missions.

7.3.4 Extension for Forwarding ‘Real’ Callback Classes

The proof-of-concept, as implemented, utilised an agent platform representing a MASHIn MASH and other agent platforms, with the agent travelling to the MASH after each hop to an untrusted agent platform. This was sufficient for satisfying the feasibility of the MASHIn SCC4MAC proposal, because the MASH first successfully received the agent from a client agent user across the network, executed the agent (with the ‘real’ callback class), then transported the agent jar without the ‘real’ callback class (i.e. just with the driver class and ‘dummy’
callback class) to the untrusted agent platforms which only saw and interpreted the agent with the ‘dummy’ callback class. When the agent returned to the MASH in between hops to the untrusted agent platforms, the MASH re-loaded the ‘real’ callback class as intended in the SCC4MAP scheme.

To minimise load on any MASH and to minimise network travel, the MASH’s preliminary task (as explained in Section 6.5) when assigned mission protection for an agent is to appropriately select checking hosts to be inserted in the agent’s itinerary. Thus, in a production MASHIn environment, the ‘real’ callback classes of an agent will be securely forwarded from the MASH to the first assigned checking host. After receiving the agent and interpreting the agent with the ‘real’ callback classes, the checking host will then securely forward the ‘real’ callback classes to the next checking host, and likewise the process repeats itself until the ‘real’ callback class and the agent is back at the MASH after its mission completion. Of course, the ‘real’ callback class may travel to a trusted agent platform as well, if there are any such agent platforms deemed trusted by the agent user\(^ {17}\). We discuss this possibility in Chapter 8.

A simple secure protocol, thus, is needed to transport the ‘real’ callback classes between MASHs, checking hosts, and trusted agent platforms in the agent’s mission. The protocol message payload will contain the agent instance unique identifier and the ‘real’ callback class. The message will be securely enveloped for the intended receipt, and stored securely in a database at the recipient site until the agent reaches it from the previous hop (i.e. an untrusted agent platform) and at which time it is readying the agent for interpretation it will load the pertinent agent ‘real’ callback class from its local database of securely forwarded ‘real’ callback classes\(^ {18}\). The message will also be digitally signed by the sending MASH/checking host/trusted agent platform for authenticating the message sender. An alternative, more efficient configuration (especially for a high-volume of messages), to this message enveloping and signing would be dedicated secure channels permanently set up between MASHs and checking hosts. The messages encapsulating forwarded ‘real’ callback classes, and other message types such as the agent TTL messages discussed in Section 6.5, would then be passed between the two TTPs using these dedicated extranet-based site-to-site Virtual Private Network (VPN) connections \[^ {172}\].

\(^ {17}\)A proposed mechanism for ascertaining stakeholder trust levels is discussed in Chapter 9.

\(^ {18}\)Once the agent is finished at the site, the securely stored ‘real’ callback class can be deleted from the local database.
7.3.5 Protecting the Privacy of Agent Platform Inputs

Generally speaking, by far the more difficult stakeholder to provide protection for in the mobile agent paradigm is the mobile agent user (and its mobile agent). This is no different in terms of protecting stakeholder privacy. We have explained our non-trivial novel mechanism, SCC4MAP, for achieving mobile agent user (code and data) privacy in the MASHIn.

To keep things fair, in terms of supporting privacy for mobile agent platform data inputs (into a mobile agent and platform negotiation), we will now discuss a trivial mechanism to accommodate this.

A mobile agent and an agent platform may provide inputs to some negotiation between themselves. Sometimes the input into this negotiation supplied by the mobile agent platform can remain in clear-text (for other platforms to view). However, in some cases the agent platform will want these inputs to be kept private from other agent platforms (and even perhaps the agent). In the MASHIn, this can be accommodated by enveloping (i.e. encrypting) the agent platform data with the appropriate MASHIn TTP’s public key.

![Figure 7.4: MASHIn method to keep agent platform inputs private from other agent platforms.](image)
Chapter 7. Protecting Mobile Agent Data and Code Privacy

Considering Figure 7.4, an agent instance has some unique identifier in the MASHIn\textsuperscript{19}. The agent travels to an arbitrary number of agent platforms. The agent platforms may wish to provide the agent with some input (e.g. from a stationary agent platform negotiation agent) that they wish to keep secret from other agent platforms (or perhaps the mobile agent itself) - remember the MASHIn is an application-independent framework. To achieve mobile agent platform privacy, the agent platform envelopes its data input and the agent’s identifier with the relevant MASHIn TTP’s public key\textsuperscript{20}. By the “relevant MASHIn TTP’s public key”, we are referring to an individual MASHIn MASH or checking host public key - not a shared public key, nor a secret threshold scheme [49, 50] among the MASHIn TTPs - though such schemes could be incorporated for adding robustness and combined distributed trust in case, for example, the targeted MASHIn TTP becomes unusable or a minimal consensus on unenveloping the data to an agent is required. Nevertheless, discussing these additional sharing and secret threshold scheme possibilities in any more detail would be going outside the scope of the main objectives of this thesis.

When the agent reaches the TTP, the agent requests the TTP (e.g. TTP Translation Stationary Agent) to unenvelope (i.e. decrypt) the data on behalf of the agent. The unique agent instance identifier included in the enveloped data signifies to the TTP Translation Stationary Agent that the (now) unenveloped data was meant for the agent and can thus be passed back to the agent. This is a precaution so that a malicious agent which encapsulates intercepted enveloped data intended for the TTP cannot lure the TTP into arbitrarily passing back data it unenvelopes to the requesting malicious agent.

If the agent platform wishes to keep its identity-to-input mapping private from an agent, then a structure of agent platform identity-to-input mappings can be built up and passed to the TTP which maintains a secret (to the agent) list of these mappings. To consider where such a scenario might be useful would involve considering something very application-specific, but we wanted to simply highlight that various forms of agent platform data privacy can also be readily supported within the MASHIn framework.

\textsuperscript{19}We discussed a possible mechanism for generating MASHIn unique agent instance identifiers in Section 6.5.

\textsuperscript{20}The TTP’s public key certificate can be encapsulated in the agent’s JAR.
7.4 SCC4MAP Comparison with Existing Strategies

In Section 7.2 we reviewed the existing mobile agent data and code privacy mechanisms located in the body of research literature, and in Table 7.1 on page 139 we provided our summary appraisal of the privacy support, practicality, effectiveness, and efficiency characteristics of the reviewed schemes. We will apply these same measures to our SCC4MAP mobile agent privacy mechanism.

In Section 7.3 we provided an extensive discussion of our novel Secure Callback Classes for Mobile Agent Privacy (SCC4MAP) mobile agent data and code privacy mechanism which fits in nicely to the overall MASHIn framework and design objectives.

Our SCC4MAP mechanism was shown to provide explicit support for both data and code mobile agent privacy. The modus operandi of our scheme falls within the ‘solve general agent platform problem case’ strategies (Section 7.2.2), not the ‘weak’ (Section 7.2.1) strategies, for achieving mobile agent privacy - which as we discussed in Section 7.2 are the more sound set of strategies from a security perspective. The SCC4MAP has a simple but effective (i.e. via ‘real’ callback class) method of handling sensitive mobile agent data and code needing privacy from untrusted agent platforms. The privacy support provided by the SCC4MAP mechanism, and MASHIn framework more generally, enable the secure management of arbitrary agent applications of any varying type and functional size.

The practicality of our SCC4MAP mechanism is very high; our proof-of-concept was successful in demonstrating the SCC4MAP mechanisms is readily deployable provided the underlying MASHIn components are in production-use. The MASHIn is not only a novel high-level security approach for protecting mobile agents, it is an admirable proposal because the primary MASHIn motivator is to bring the two mistrusting stakeholders into an environment more conducive to interaction - a perspective and direction others have largely missed when studying the viability of practical Internet mobile agents. Projecting the use of the MASHIn means there are genuine independent TTPs (i.e. MASHs and checking hosts) having no application or stakeholder bias. We even showed how the MASHIn can be trivially utilised to support agent platform privacy (from other agent platforms and even agents) in Section 7.3.5.
Our SCC4MAP mechanism is a very highly effective mobile agent data and code strategy since no mobile agent data or code deemed sensitive by the agent user is made accessible (and thus visible) to untrusted agent platforms. Furthermore, in terms of robustness, the MASHIn’s structured monitoring support for a deployed agent means the agent’s chances of mission completion remain high should something go wrong (e.g. an agent platform denies service) - a factor largely neglected by existing agent security approaches, not just with privacy-specific approaches.

The SCC4MAP mechanism in terms of efficiency is quite reasonable. There is an additional network message per untrusted agent platform hop with the incorporation of MASHIn TTPs for forwarding on ‘real’ callback. Considering the high quality of mobile agent privacy provided by the SCC4MAP mechanism, this itinerary-time overhead is very acceptable given the MASHIn fulfills both key goals for Internet mobile agents - security and autonomy. Network bandwidth and speeds are multitudes greater than when mobile agents were first promoted in the early-mid 1990s. Nevertheless, the proposed incorporation of strategically-selected checking hosts (having a reasonable processing load and in close proximity to the pertinent untrusted agent platforms) by the MASH, and dedicated extranet site-to-site VPNs between the MASHIn TTPs helps to lessen the additional network processing costs of incorporating TTPs in the MASHIn distributed environment.

Overall, we believe our SCC4MAP mechanism represents favourable qualities in terms of the analysed properties of privacy support, practicality, effectiveness, and efficiency as a mobile agent privacy strategy. The SCC4MAP approach is an application-independent, agent system language/environment-independent, practical and effective novel mechanism for achieving mobile agent privacy. It is our strong opinion that our SCC4MAP mechanism stands out as a leading approach to achieving mobile agent data and code privacy in light of existing mobile agent privacy mechanisms.

7.5 Summary

In the previous chapter we provided an overview of the fundamental components in the MASHIn strategy for bringing the two major Internet mobile agent paradigm stakeholders into an environment more conducive to safe interaction,
and thus a practical business relationship.

The central abstraction in the MASHIn are TTP MASHs which provide, or at least act as the pivotal entity in making viable a number of application-independent security services for both stakeholders, mobile agent users and mobile agent platform owners.

In this chapter we took a look at how the solid inherent robust design of the MASHIn facilitates high-quality mobile agent data and code privacy. In Section 7.1 we pointed out the flexible nature of mobile agent data and code privacy that we wished to achieve. In Section 7.2 we reviewed the existing mobile agent privacy mechanisms found in the relevant body of research literature, and provided an appraisal of their pros and cons - with a general conclusion that they lacked necessary flexibility and robustness requirements for our needs. Our aim was focused on not enabling potentially malicious agent platforms accessibility to sensitive agent data and code, thus avoiding the possibility of that data being compromised (reverse engineered, decrypted, weakened by collaboration attacks, etc.). We then discussed our novel approach (titled Secure Callback Classes for Mobile Agent Privacy, SCC4MAP) for achieving secure mobile agent data and code privacy in Section 7.3, and in Section 7.4 we provided a comparative discussion of our SCC4MAP approach in relation to the existing reviewed mobile agent data privacy approaches - noting good results on the analysed criterion.

In the next chapter (i.e. Chapter 8) we look at the important property of maintaining execution integrity as a mobile agent hops around during its mission including visits to untrusted agent platforms, and specifically how this can be most effectively handled within the MASHIn framework.

Then in Chapter 9 we discuss a proposed method of maintaining timely personalised trust levels of other stakeholders, based on MASHIn-TTP first hand observations as inputs into reputation appraisals. The MASHIn reputation model (along with the MASHIn’s execution integrity model) is useful for a number of reasons, not the least which is that otherwise unknown (and by default, thus, untrusted) stakeholders can transparently form new working relationships.

Finally, in Chapter 10 we discuss some design considerations for protecting MASHs, the lynch-pin MASHIn TTP entity on which much safeguarding of stakeholder security (and thus business-continuity) interests is dependent.
Chapter 8

Protecting the Execution Integrity of Mobile Agents

If the truth doesn’t save us, what does that say about us?

Lois McMaster Bujold

In the previous chapter we discussed how mobile agent data and code privacy can be achieved, particularly in the context of the MASHIn. With respect to that matter, we showed that our Secure Callback Classes for Mobile Agent Privacy (SCC4MAP) approach fitted in well within the MASHIn framework, and ranked favourably when compared to other mobile agent data and code privacy approaches analysed from the existing body of research literature.

In this chapter we focus on mobile agent execution integrity, a very important property - particularly to the confidence an agent user can place in the results of its deployed autonomous mobile agents (see Section 3.4 for discussion on a number of threats to the correct results of deployed mobile agents). We start in Section 8.1 by expanding on what is involved in providing a sound solution to mobile agent execution integrity. To assist in this matter we consider a non-trivial mobile agent application, and we produce a number of objective qualities that we would expect from a solid mobile agent execution integrity mechanism. In Section 8.2 we review the major existing approaches in the body of research literature supporting mobile agent execution integrity, and analyse them in light
of objective properties listed and discussed in Section 8.1. In Section 8.3 we discuss in some detail our application-independent execution integrity approach for the MASHIn, building and improving on the most promising of the existing execution integrity approaches reviewed in Section 8.2. This MASHIn execution integrity approach, which we have titled Real-time Execution Checking and Deterrent Against Misbehaviour (RECDAM), is then compared to the existing execution integrity approaches in Section 8.4. Finally in Section 8.5 we summarise the chapter and the usefulness of the RECDAM approach to supporting the MASHIn’s overall objectives.

This chapter includes and builds on material found in peer-reviewed accepted papers [i]-[iv] listed on page xxi.

8.1 Execution Integrity Objectives

In this section we will highlight the properties that we feel are necessary in a solid execution integrity scheme for mobile agents. To assist in explaining these properties, in Section 8.1.1 we discuss a non-trivial mobile agent application, and in Section 8.1.2 we list the properties - referring to the application example for clarification where needed - that we believe are vital in providing a robust execution integrity scheme for application-independent mobile agents.

8.1.1 Quality Research Paper/Author Finder Mobile Agent Application

We all know that finding quality research papers, especially fresh work, in a research field is often a time-consuming and non-trivial endeavour - for example - due to subscription costs to journals, Internet connectivity problems, copyright restrictions, language barriers, and so on.

Assuming a ubiquitous secure Internet framework, one could imagine many applications of mobile agents in providing researchers more transparent access to quality research papers and enabling automated collaboration with other researchers in a timely manner. We will now discuss one possible mobile agent application which assists in quality research paper/author retrieval via synchronous collaboration with respected individuals in a research domain.

Imagine a mobile agent whose mission is to target the most promising new
research papers in a particular field, or perhaps the most promising talented researchers in a particular field. This could be, for example, to assist in forming a special issue “Hot Topics in X, Y, Z” for some particular journal, or to instigate possible recruitment of talented research investigators for a crack new research institute team.

The mobile agent can collaborate with other agents - either stationary agents on agent platforms it visits, or other mobile agents via inter-agent messaging. Interacting stationary/mobile agents input into the agent’s mission search their list of top (some number) $n$ research papers, as requested, in the particular topic field. A rating is also supplied by the interacting agent. This rating could be based on widely varying criteria for example originality, technical application, quality of writing, and potential viability for future research - to name just a few.

Consideration of the recommender’s advice is weighted by the mobile agent on a confidence level in the critiquing principal’s aptitude; the agent user will often want to keep these confidence levels private from agent platforms (more specifically the recommenders) offering the advice.

The agent is initially dispatched to a static number of agent platforms, with each recommender (on an agent platform) having a pre-assigned aptitude confidence level. The agent’s mission is not complete until it has attained $n$ authors or papers having amassed a minimum sum of weighted scores.

If at the end of the static number of agent platform visits, the agent’s mission goal has not been reached, it can visit other recommender sites - for example those associate recommenders highly commended by visited recommender sites. These new recommenders are assigned a fraction of the recommending sites’ aptitude confidence levels, which should also be kept private.

Eventually, it is likely that the agent will have completed its mission and the results can be returned to the agent’s user. Even if it has not reached all of the required quality papers/authors finding, the mobile agent will have (ideally) exhausted all recursive possibilities in a reliable, safe manner.

8.1.2 Properties of Interest

We define a competent mobile agent execution integrity scheme as not only providing comprehensive protection for the mobile agent’s data and code integrity, but also irrefutably linking the agent user and agent platforms to their inputs into any interaction via the agent’s execution.
Furthermore, an execution integrity scheme for mobile agents is ineffective without a readily accommodating means of recovering from a hijacked agent session on an agent platform. In more traditional distributed systems, the distinction between the desired properties of integrity per se and availability per se is more clear. However, in mobile agent systems the “playing field” for mobile agents is a series of agent platforms. There must be a way to irrefutably link both the agent and agent platform inputs into an interaction at least up until a point of failure in the agent’s itinerary. For example, if there is a significant disruption to the agent’s mission at or around the fifth agent platform in an agent’s itinerary, the results and interactions of the agent at agent platforms 1-4 inclusive must not be lost. Surprisingly, this aspect is not accommodated in many existing execution integrity or denial-of-service schemes for mobile agents, with drastic consequences likely since there is no accountability up until the point-of-failure, meaning that these interactions are left in a state of jeopardy (for example, should they be rolled-back or agent platform results/service remuneration somehow fairly discarded/handled?) and the responsibility for the failure cannot be attributed to an entity (was the mobile agent responsible, or was the agent platform responsible?). Besides not losing these previous work-flows, the scheme should also provide a means around the point of disruption in the mobile agent’s itinerary.

We elaborate, below, on the specific qualities that we view as essential in a comprehensive execution integrity package for the Internet mobile agent paradigm:

- **Application-Independent**: This one should be quite clear; if the execution integrity scheme is not mobile agent application-independent, then only a subset of agent applications can benefit, limiting the usefulness of the scheme and further hampering the interoperability and practicality of safe methods for enabling ubiquitous Internet mobile agents. Thus, an application-independent execution integrity scheme would be able to transparently protect window-shopping mobile agents, purchase-shopping mobile agents, research paper/author finder mobile agents, and any other utilisation of mobile agents.

- **Dynamic Itinerary Changes Possible**: The execution integrity scheme must be capable of handling dynamic changes to an agent’s itinerary, otherwise all agents would be confined to pre-defined statically-defined mission itineraries - limiting the autonomous benefits of mobile agents. For example, in the research paper/author finder application (in Section 8.1.1), the mobile agent’s
8.1. Execution Integrity Objectives

goal would be left unfilled in the case that the base agent platforms to visit have not provided a significant-enough count of quality research papers/authors.

- **Real-Time In-Mission Checking Possible:** The execution integrity scheme must be capable of detecting breaches of integrity in an agent and claimed stakeholder inputs in “real-time in-mission”, i.e. as the agent is in the midst of its mission, before and after the visit to an agent platform. Otherwise, once again, the damage done can leave agent platforms in an ambiguous state (how to appropriately recover from a post-mission-completion-detected breach of integrity?) with regard to issues of rollback and remuneration for services delivered. If real-time in-mission checking is supported, the damages from a breach of execution integrity can be localised, and the responsible entity more easily identified.

- **Unconditional Autonomous Checking Post-Mission Possible:** Regardless of whether real-time in-mission checking is possible, in some instances post-mission checking may be necessary; for (example) reasons of efficiency or legal enactment, it may be necessary to only perform execution checks post-mission, or in addition to real-time in-mission checking. A high-standard execution checking scheme for mobile agents must be capable of performing post-mission autonomous execution checks. This means the scheme should not rely on the agent user (or home platform) to perform the execution checking, and the checking should not be triggered only on the suspicion of the agent user.

- **Resistant Against Stakeholder Denial-of-Service:** As mentioned in the introductory discussion to this list, for the reasons given there, a reliable mobile agent execution integrity scheme (and, thus, the mobile agent’s mission results prior to, and in future of, the disruption) should not be rendered useless if a stakeholder either deliberately or unintentionally causes a denial-of-service to the mobile agent.

- **Agent Platform Collaboration Attack Resistant:** An execution integrity scheme must not be susceptible to collaboration attacks\(^1\) from agent platforms breaking the execution integrity of a mobile agent.

\(^1\)Some genuine agent platform collaboration attack threats against existing execution integrity mechanisms are discussed in Section 8.2.
• **Real Punishment Deterrent:** Catching breaches to execution integrity from either stakeholder is only part of the battle. In fact, if there is not an automated system mechanism to punish stakeholders for their discrepant behaviour, then the perpetrators may feel confident in continuing with their reckless/malicious behaviours. Ideally, the punishment should not just be a post-malicious-reactionary event for breaches but should also act as a significant real deterrent against stakeholders breaching execution integrity in the first place.

• **Effective:** The execution integrity scheme must be easy to manage and transparent to incorporate, and not simply be a fantastic theoretical proposal which has limited/non-robust utility in generic mobile agent applications.

• **Efficient:** Given all of the above features are present, the execution integrity scheme should not place unreasonably inefficient procedures (neither on an agent platform, nor in terms of network messages) in meeting its objectives.

In the next section we provide an overview of the existing strategies found in the research body of literature supporting execution integrity, and we analyse how they rate against the qualities that we believe are paramount, as outlined above, in the formation of a robust and flexible execution integrity scheme for mobile agents.

### 8.2 Review of Existing Strategies

In this section we review the range of execution integrity strategies for mobile agents from the existing body of research literature. In Section 8.2.1 we discuss partial result encapsulation, in Section 8.2.2 holographic proofs, in Section 8.2.3 execution tracing, and we discuss the reference states approach in Section 8.2.4. Finally, in Section 8.2.5, we analyse these execution integrity approaches for the presence of qualities (discussed in Section 8.1.2) that we deem necessary in a solid execution integrity scheme for mobile agents.

#### 8.2.1 Partial Result Encapsulation

Yee [237] proposes partial result authentication codes (PRACs) for encapsulating the application-independent partial results of an agent’s actions at agent plat-
forms. Depending on the specific implementation employed, these results can be either verified at intermediate points in the agent’s itinerary or post-mission on the agent’s return to its point-of-origin. PRACs are cryptographic checksums formed using secret key cryptography, and have some similar properties to message authentication codes (MACs). The technique requires the agent and its originator to maintain or incrementally generate a list of secret keys used in the PRAC computation. Once a key is applied to encapsulate the information collected, the agent destroys it before moving onto the next platform, guaranteeing forward integrity\(^2\). However, in this method of operation, only the agent user can verify the results, since no other copies of the secret key remain. Alternatively, public key cryptography and digital signatures can be employed in lieu of the secret key techniques. This alternative has the benefit that authentication of the results can be made a publicly-verifiable process at any agent platform in the agent’s itinerary, while maintaining forward integrity [96].

In different scenarios, the agent can itself perform the encapsulation (of the partial results), while in others the agent platform is prescribed to perform the encapsulation. To meet certain security requirements such as integrity and accountability, and privacy of the agent platform inputs, the partial result encapsulation mechanism makes use of different cryptographic primitives such as encryption, digital signatures, authentication codes, and hash functions [5, 96].

Three general techniques to encapsulate the partial results of mobile agents can be supported [96]:

- Provide the agent with a means for encapsulating the partial results: This can be applied independently in the design of an agent, regardless of the capabilities of the agent platform or supporting infrastructure [237, 96].

- Rely on the encapsulation capabilities of the agent platform: Rather than relying on the public-key of the agent user to encapsulate partial results, all agent platforms (in the agent’s itinerary) sign the partial results with their private key [176, 35].

- Rely on a TTP to time-stamp a digital fingerprint of the partial results: A digital time-stamp from a TTP can enable publicly-verifiable partial results [237].

\(^2\)The forward integrity aims to ensure that if one of the agent platforms visited is malicious, the previous set of partial results accumulated by the agent remains valid.
A serious vulnerability arises in the PRAC approach if a malicious agent platform retains copies of the original keys or key generating functions of an agent. If the agent revisits that offending agent platform or visits another agent platform maliciously collaborating with it, a previous partial result entry or series of entries could be modified leaving the agent user unawares.

A application-dependent partial result encapsulation technique developed by Young and Yung allows small amounts of data (only) to be encrypted yielding compact results [239]. Though the main purpose of sliding encryption is agent platform input privacy (and a secondary benefit is making an agent’s travels throughout a network difficult to trace), an additional integrity measure could be applied before the encryption process [96].

Other researchers have incrementally improved on the cryptographic techniques/properties involved in Yee’s PRAC scheme for their varied application-specific or agent system-specific purposes [109, 167, 233, 241]. Most of these enhancements are geared towards further protecting shopping application mobile agents, and address base flaws (such as susceptibility to truncation or collaboration attacks) which should have been avoided in not only protecting shopping application mobile agents geared for use in e-commerce, but any solid execution integrity approach geared for generic mobile agent applications.

### 8.2.2 Holographic Proofs

The aim of holographic proofs, standalone or used with computationally sound proofs, is to show that a mobile agent’s computation at agent platforms ran correctly (i.e. as intended by the agent user’s program), without the overhead of verifying a complete execution trace of the agent’s mission [237]. A trace \( T_p \) of the execution of program \( p \) is composed of a sequence of pairs \( \langle n, s \rangle \), where \( n \) represents a unique identifier of a statement, and \( s \) is an agent platform digital signature. Traces are used for the post-mortem analysis of data, as a basis of program verification, to check the agent program against a supposed history of its execution [213].

The basis of a holographic proof is very theoretical, and whilst in principle it is helpful it is in practice prohibitively expensive. Yee explains holographic proofs as follows [237]:

Call the program \( x \). Let \( y \) denote an execution trace. Define the predicate ‘\( (x,y) \)’ to be true if this trace is correct (corresponding to
running \( x \)) and false otherwise. The agent platform does not want to send \( y \). But it can encode \( y \) as a holographic proof \( y \). This holographic proof \( y \) has the property that one needs a few bits of \( y \) to check that \( (x, y) \) is true. It is tempting from this to think the agent platform can just transmit a few bits but this does not work. The model necessary for holographic proofs is that the verifier have available a fixed “committed” proof string \( y \) that he can access at will. We will pick a few random positions here and check something. So there is no choice but to transmit \( y \) in its entirety. We will not save bandwidth (of the submitted execution trace), but we will gain something with the verification of the proof being faster.

By extending Yee’s above (Holographic Proofs) proposal Biehl, Meyer, and Wetzel claim to show that substantial bandwidth savings of the execution trace can be made [19]. Whilst they acknowledge the basic holographic proof theory is a long way off a practical and efficient solution to execution integrity for mobile agents, they nevertheless report that “short” proofs of the correct execution of a mobile agent are feasible.

The idea of the computationally sound proof approach [129] is to offer a “computational” guarantee with the features of “easy to find” and “easy to verify”, which means that false statements either do not have any computationally sound proofs, or their “proofs” are practically impossible to find [233]. Yee suggests that applying the theory behind computationally sound proofs with holographic proofs for limited functional mobile agents of interest may be a promising direction to further investigate [237].

### 8.2.3 Execution Tracing

The execution tracing [212, 213] mechanism developed by Vigna is an application-independent execution checking mechanism for detecting unauthorised modifications of an agent through the faithful recording of the agent’s behaviour during its execution on each agent platform. Each agent platform creates and retains a non-repudiable log (i.e. the trace) of the operations performed while the agent is resident there, and submits a cryptographic hash of the trace upon conclusion to the agent user or a TTP as a trace summary or fingerprint [96, 5].

A trace point is, typically, recorded each time the state of the program is modified using information from the external environment. As an example, the
statement \texttt{read(\textit{x})} modifies the variable \textit{x} with data received from the executing agent platform environment. A trace point recorded for this event will contain a unique identifier of the statement executed and a signature containing the result of the statement execution. Normally, the trace log is stored on the executing agent platform for a long period of time, and is only returned to the agent user if requested - triggered on the suspicion of the agent user [40]. Vigna’s execution tracing protocols include the possibility of transmitting the traces to a TTP to retain a sequence of trace summaries for the agent’s entire itinerary. Whether a TTP is employed or not, the execution tracing mechanism also affords some protection to an agent platform if an agent (user) or another agent platform accuses it of mishandling the agent during its stay there [96, 212].

The agent’s user is responsible for validating the execution of the agent by comparing the fingerprint of the reproduced out-of-band agent execution trace/s against the fingerprint of the trace/s that are originally supplied by the suspicious platform/s [5, 212].

Regardless of whether the traces are submitted for storage at a TTP, Vigna’s execution tracing has a number of drawbacks including the size and number of traces to be attained, the fact the detection process is triggered only occasionally (and only on the suspicion of the agent user), and there is no support for multi-threaded agents [96, 212, 5]. Furthermore, a significant problem with execution tracing is that the scheme relies on the good-will of the agent platform to maintain the execution traces, which reduces its robustness.

Tan and Moreau [194] have improved on the execution tracing mechanism devised by Vigna. Their extended execution tracing mechanism mandates the incorporation of a \textit{verification server} for submission of execution traces to validate the claimed agent platform output before the agent is permitted to migrate to the next agent platform. Thus, Tan and Moreau’s extended execution tracing mechanism improves on Vigna’s initial strategy by dictating real-time in-mission execution integrity checking. They also provide a possible mechanism by which verified execution traces can be incorporated in tracking agent platforms who have a reputable history of performing execution of agents per their intended specification [193]. Tan and Moreau [194] suggest that the extended execution tracing should incorporate a fault tolerant scheme using time-outs to assist in detecting and working around some denial-of-service attacks or busy agent platforms, as we have incorporated in the MASHIn design (detailed in Section 6.5).
Contrary to claims made by Alfalayleh and Brankovic [5], the extended execution tracing mechanism utilising a verification server does not suffer (to the same degree) from the need to retain a potentially large size and number of traces as in Vigna’s [212] execution tracing mechanism; Tan and Moreau [194] dictate that only discrepancies (i.e. breaches) in traces are retained by the verification server, for possible future evidence in claims made by the agent user. In the original execution tracing mechanism [212] agent platforms had to retain traces because the checking was only spasmodically checked by the agent user on their suspicion that something went amiss with the formation of his/her agent’s mission results; but, in the extended version by Tan and Moreau [194] the checking is done by verification servers real-time in-mission. A problem arises, though, if the agent user disputes the execution of its agent and one or more of the verification servers has deleted copies of the checked execution traces (because they were of the opinion that the agent’s execution by their associated agent platform was valid). The agent user, thus, has no evidence to base a further investigation of the execution integrity of its agent’s mission.

However Alfalayleh and Brankovic [5] quite rightly point out that each agent platform chooses a verification server and it is concluded that this might encourage and facilitate a possible malicious collaboration between an agent platform and the verification server it chooses to employ [5]. We are unsure why it is not the agent user (rather than each agent platform) [194] who dictates the verification servers to utilise. The extended execution tracing mechanism would clearly be a much stronger method of ensuring the execution integrity of mobile agents if the agent user (or, even better, some neutral TTP proxy-server) chose the verification servers before an agent is dispatched to the first agent platform in its itinerary.

8.2.4 Reference States

Hohl [85] defines a reference state to be the combination of the variable parts (i.e. the state) of a mobile agent executed by an agent platform showing reference behaviour, with a “reference” agent platform being one which performs the agent computation correctly. In attempting to overcome weaknesses in other schemes which embody reference state characteristics [134, 59, 212, 237] including execution tracing and holographic proofs, Hohl has come up with a more powerful reference states mechanism.
The major improvements made were to have the execution checking performed after every agent platform - not just after a suspicion is raised, and to perform the execution checking autonomously (i.e. not instigated and performed by the agent user at the home agent platform, as this breaks one of the key advantages of mobile agents - asynchronous execution) [85]. The agent user’s home platform may not have the necessary computing power and/or security algorithms to perform the checking. An agent user should just be interested in, and provided with, securely transported (e.g. via PDA, mobile phone, s/mime email, etc.) correct results of its agent mission. In fact, the agent user may not even have a home agent platform, and even if it does that platform may be disconnected from the network - neither should prevent the safe calculation and return of the agent’s results to its user.

Hohl’s reference states protocol [86, 85], in simplified terms, provides a series of unconditional steps which encapsulate re-computing the execution session of an agent at one agent platform on the next agent platform using the inputs to the agent from that preceding agent platform session. In an arbitrary $n$-agent-platform itinerary, agent platform 2 would be checking the claimed resulting state received from agent platform 1 using agent platform 1’s initial state (of the agent) and agent platform 1’s input (which are sent in a secure message from agent platform 1 to 2). If the claimed result matches its verified state, agent platform 2 proceeds to do its own servicing of the agent, otherwise it complains and stops processing the agent. Then, (given reference state has not failed) after the agent migrates to agent platform 3, agent platform 3 will check agent platform 2’s claimed resulting state using the initial state of the agent to agent platform 2 and agent platform 2’s input. And, so on.

There are a couple of slight differences in the design of Hohl’s reference states protocol to that of the simplified explanation given above, including optimisation for trusted and non-trusted agent platforms, but the thrust of the above explanation holds true in the main part. As would be expected, the cost of using the (non-optimised) protocol basically doubles that of a non-checked itinerary run, because for each agent platform execution session there is an execution check on the next agent platform [85].

Hohl, in another paper on his reference states scheme, provides a Java implementation framework for a range of checking mechanisms using reference states, allowing the agent programmer to decide the check mechanism an agent platform is mandated to employ. The framework also directly supports functionality like
digitally signing claimed initial and resulting states, and declaring agent platform inputs [86].

Hohl has readily identified two major weaknesses with his reference states approach outlined above. Firstly, collaboration attacks of two consecutive agent platforms cannot be detected by future agent platforms. Secondly, input to an original execution cannot be held secret from the checking - possibly competitive - agent platform (by the very nature of the protocol requiring the agent platform to send the next agent platform its input encapsulated in the agent’s initial state so that the next agent platform can verify the claimed resulting state against the calculated state it checks) [85].

We come back to Hohl’s reference states approach in Section 8.3 when we appropriately adapt and improve (including overcoming the two noted problems) on this scheme to form the most robust execution integrity mechanism for mobile agents that we are aware of. As a bonus, our execution integrity scheme fits nicely within the framework of the MASHIn.

8.2.5 Summary of Existing Approaches to Providing Execution Integrity

In Table 8.1 we provide a summary analysis of the robustness (based on the qualities discussed in Section 8.1.2) of the execution integrity approaches for mobile agents reviewed in this section.

The Partial Result Encapsulation (PRE), Execution Tracing (ET), and Reference States (RS) approaches are mobile agent application-independent schemes, meaning they can be reasonably applied to agents of arbitrary application-type and code size. The direct construction of Holographic Proofs (HP) for arbitrary agents is not feasible at this time.

Dynamic itinerary changes are possible in all four reviewed schemes, with the caveat that it is not possible in all implementation varieties of the PRE concept.

Real-time in-mission checking is supported fully only in the RS scheme. However, in some implementation varieties of PRE it is supported, and in the extended ET scheme by Tan and Moreau [194] it is directly supported.

Unconditional post-mission autonomous checking is not supported by any of the schemes. The first three schemes rely on the agent user (or their agent platform) to conduct post mission checks, thus breaking the autonomous property we desire. The RS scheme does not support post-mission checking directly, and
it would only be possible to extend the scheme by introducing a TTP receiving reference states messages (otherwise the agent user/home platform would have to perform the checks, thus breaking the autonomous checking property).

None of the schemes, except Tan and Moreau’s extended ET [194] scheme, provide resistance against denial-of-service or denial-of-execution by stakeholders preventing correct interpretation of an agent on an agent platform (session). Moreover, the results from agent platform sessions in the agent’s itinerary preceding the point-of-failure are often lost or left in a state of inconsistency.

All of the schemes are directly vulnerable to agent platform collaboration attacks, and none of them provide any automated system punishment mechanism for deterring future attacks.

The PRE scheme is only medium effective because its implementations often rely on the agent user/home platform to perform the execution integrity checks. Cryptographic key generation and/or storage management issues and agent platform collaboration attacks are also limiting factors. Nevertheless, the PRE scheme scores admirably in terms of efficiency.

The HP scheme rates poorly in terms of both effectiveness and efficiency, and cannot be seen to be a viable solution to execution integrity of arbitrary mobile agents in the near future.

The ET scheme is only medium-effective because the cryptographic traces to generate and manage would be large for both sets of stakeholders. However, the extended ET scheme lessens this load, but introduces its own set of problems (particularly, vulnerability to malicious collaboration between an agent platform and verification server). The number and size of the cryptographic messages are also a concern to the scheme’s efficiency.

The RS scheme is limited in its design, leaving it - for example - highly vulnerable to agent platform collaboration attacks. However, the scheme’s efficiency is admirable, and the RS property of real-time in-mission checking is very admirable.

In conclusion, we find none of the reviewed schemes to be sufficiently robust in satisfying our requirements for a solid mobile agent execution integrity scheme. In the next section we discuss our MASHIn execution integrity scheme, starting first by investigating Hohl’s RS scheme in more detail. We particularly build and improve on his scheme for two main reasons. Firstly, we favour the scheme’s very admirable qualities of real-time in-mission execution integrity checking and
performing these security checks completely autonomous of the agent’s user and home agent platform. And, secondly, we can workaround a number of the limitations associated with the RS scheme by appropriately adapting it to work within the MASHIn framework; at the same time increasing the scheme’s effectiveness.
### Table 8.1: Appraisal of mechanisms for providing robust mobile agent execution integrity.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>AI</th>
<th>DIP</th>
<th>RTC</th>
<th>UPMAC</th>
<th>SDoSR</th>
<th>APCAR</th>
<th>RPD</th>
<th>Effectiveness</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial Result</td>
<td>✓</td>
<td>%</td>
<td>%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Medium; often relies on agent user/home platform to perform checks</td>
<td>Medium-Good</td>
</tr>
<tr>
<td>Encapsulation (PRE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Holographic Proofs (HP)</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Impractical now, and likely to remain so</td>
<td>Poor</td>
</tr>
<tr>
<td>Execution Tracing (ET)</td>
<td>✓</td>
<td>✓</td>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td>Medium; but the cryptographic trace management is problematic</td>
<td>Poor-Medium</td>
</tr>
<tr>
<td>Reference States (RS)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Poor-medium, because of the collaboration attack and input privacy problems</td>
<td>Medium-Good</td>
</tr>
</tbody>
</table>

**Table Abbreviations:** AI = Application-Independent; DIP = Dynamic Itinerary Possible; RTC = Real-Time Checking; UPMAC = Unconditional Post-Mission Autonomous Checking; SDoSR = Stakeholder Denial-of-Service Resistant; APCAR = Agent Platform Collaboration Attack Resistant; RPD = Real Punishment Deterrent.

**Table Symbol Key:** “✓” = feature supported; “+” = additional supported feature in the extended execution tracing method; “%” = supported in some forms of scheme.
8.3 Real-time Execution Checking and Deterrent Against Misbehaviour

We highlighted in Section 8.2.5 that none of the existing execution integrity strategies came close to meeting our requirements for a solid execution integrity mechanism for mobile agents. In this section we will discuss our MASHIn execution integrity mechanism which satisfies all of our requirements (outlined in Section 8.1.2) for a robust execution integrity mechanism for mobile agents. We will discuss how one can go from the general unprotected execution integrity model for mobile agents, depicted in Figure 8.1, to a solid execution integrity scheme.

The unprotected execution integrity model for mobile agents, represented in Figure 8.1, involves transporting agent code and its state\(^3\) from a hosting agent platform (once the agent is finished executing there) to the next agent platform. The resulting code/state data values transported to the next agent platform are not digitally signed by the sending agent platform, so there is no possibility of

---

\(^3\)In Java-based mobile agents, only mobile agent code can migrate, not pure state (i.e. the stack/heap) data per se. The state of an agent in Java-based mobile agents must therefore be stored and re-simulated via class instance variables encapsulated in serialised byte format, or external values such as those that might be stored within the JAR-file encapsulating the agent.
ensuring non-repudiation of these values.

In Section 8.3.1 we discuss Hohl’s reference states protocol which we adapt and incorporate within the MASHIn framework. Hohl’s scheme provides admirable objectives for the execution integrity checking of mobile agents but suffers from significant design limitations. Section 8.3.2 provides an overview of the pertinent changes, and improvements, that we made to Hohl’s reference states protocol so that it can work appropriately within our Real-time Execution Checking and Deterrent Against Misbehaviour (RECDAM) execution integrity strategy. The three possible modes of our MASHIn RECDAM execution integrity scheme are then presented in Section 8.3.3, along with a simple proposed fee-charge model for utilising the RECDAM service. Finally, in Section 8.3.4 we outline how the MASHIn can flexibly respond to execution integrity events, and how the output of MASHIn RECDAM execution integrity can be partially used in the determination of entity trust confidence levels (discussed in some detail in Chapter 9).

8.3.1 Hohl’s Reference States Protocol

8.3.1.1 Introductory Notes on Hohl’s Scheme

Hohl’s reference states strategy [85] overcomes the two shortcomings in the unprotected execution integrity model for mobile agents just outlined. Figure 8.2 shows the basic checking process that is mandated before an agent is interpreted as expected on a receiving agent platform. The receiving agent platform is first required to re-execute and check the execution of the agent from the previous agent platform. It can do this if the initial agent code/state and any inputs into the agent’s execution on the previous agent platform are irrefutably transported, in addition to the resulting code/state. The agent is only then interpreted as expected on the receiving agent platform if the claimed resulting state from the previous agent platform is verified by the execution checking process on the current (i.e. receiving) agent platform.

In contrast to Vigna’s execution tracing mechanism [212] (a high-level overview was presented in Section 8.2.3), the checking process is not done on the home agent platform - but immediately on the next agent platform in the agent’s itinerary. Autonomous execution integrity checking is important because the agent user may have disconnected from the network (i.e. mobile device not in a hot spot, or be running short on battery-life) or be unable to perform the nec-
8.3. Real-time Execution Checking and Deterrent Against Misbehaviour

Figure 8.2: Re-checking in Hohl’s reference states scheme, as the first step on an agent platform.

necessary cryptographic processing (due to hardware and/or software limitations). From the agent user’s perspective, they simply want a reliable summary of the agent’s security-ensured mission results. These mission results could be securely communicated to the agent user in a variety of ways (e.g. s/mime email, secured SMS mobile phone message, certified or discreetly packaged postal mail) with the important fact that the agent user may not have a home agent platform processing environment.

The input data transported in the reference states protocol contains all input including third-party agent communication messages, and data that was produced by the agent platform using its local resources like databases, random generators, or the system clock. A protocol is needed to ensure that the information transported is non-repudiable and protected for integrity, and the checking processes are not abused - for example by the suppression or insertion of third party communication from a malicious agent platform [85].

Before we take a detailed look at Hohl’s reference states protocol, two entity terms need to be defined in context of the reference states scheme. A trusted agent platform is one known in advance (by the agent user) to not attack the

---

4In some cases, it is possible that there is no input to the agent from an agent platform.
agent and is willing to execute checks for the agent. An untrusted agent platform is one in which the agent user is uncertain whether it will attack the agent. Untrusted agent platforms are visited since the agent wants to have something executed there that relates to the agent’s mission. Furthermore, untrusted agent platforms are willing to perform checks and follow the reference states protocol if this work is required to clear them of responsibility for any harm that may become an attacked agent. However, as Hohl rightly points out, untrusted agent platforms may even try to attack an agent while in the checking process [85].

Hohl defined non-optimised and optimised versions of his reference states protocol [85]. The non-optimised version is appropriate when only the first and last agent platform are trusted, for example the agent starts and ends at its home agent platform. However, intermediate agent platforms may be trusted by the agent user. Using the first protocol in these circumstances is not optimal. Thus, in the optimised version the computations of a trusted agent platform do not require checking on the next agent platform, because - by definition - they do not attack. Because the optimised version is sufficiently flexible to accommodate both case scenarios, we will not touch on Hohl’s non-optimised reference states protocol again.

There are actually three protocols in Hohl’s optimised reference states strategy. The first protocol is straightforward to follow, and the third protocol is only slightly different from the second protocol. Which protocol is enacted depends on the location of the agent as follows:

- If the agent is located on the first agent platform (i.e. the home agent platform), the protocol discussed in Section 8.3.1.2 is applied.
- If the agent is on an untrusted agent platform, the protocol discussed in Section 8.3.1.3 is applied.
- If the agent is on a trusted agent platform, the protocol discussed in Section 8.3.1.4 is applied.

We will now, in turn, discuss each of these protocols. Please note that when there is an instruction in the following protocols of the form $\text{sign}_x(y)$, this represents the digital signature of agent platform $x$ on message $y$ - it does not include message $y$. 
8.3.1.2 First Agent Platform Protocol

Figure 8.3 lists Hohl’s reference states protocol steps for the first agent platform (i.e. the home agent platform from which the mobile agent is launched), which is obviously trusted by the agent user.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Compute state_2</td>
</tr>
<tr>
<td>1.2</td>
<td>Transfer agent code to next agent platform if needed</td>
</tr>
<tr>
<td>1.3</td>
<td>Add sign_1(state_2) to message</td>
</tr>
<tr>
<td>1.4</td>
<td>Add state_2 to message</td>
</tr>
<tr>
<td>1.5</td>
<td>Sign message and send it to the next agent platform</td>
</tr>
</tbody>
</table>

The resulting state from executing the agent (i.e. state_2) is calculated. The agent code is transported to the next agent platform, as needed. Then a message is built up with the resulting state calculated in step 1.1, and a signature of this raw state data. The message is digitally signed by the home agent platform and sent to the next (i.e. first remote) agent platform in the agent’s itinerary.

Depending on whether the next agent platform is untrusted or trusted by the agent user, the protocol in Section 8.4 or in Section 8.5 will be invoked respectively.

8.3.1.3 Untrusted Agent Platform Protocol

Figure 8.4 lists Hohl’s reference states protocol steps for an arbitrary (in terms of the position in an agent’s itinerary) untrusted agent platform.

The first step is a standard message authentication and integrity check. If the signature is broken the agent platform can ask the previous agent platform to send a verifiable message. A broken signature here does not reflect that the previous agent platform attacked the agent. If no properly signed message is subsequently transported and received the protocol processing stops, otherwise the protocol continues.

The resulting state (state_{i+1}) from the previous agent platform is taken from the message, as well as the signature of the previous agent platform signing this resulting state - and the signature is verified. Hohl in [85] states that this might seem to be doing double the work as the whole message, also containing this raw
resulting state was already signed by the previous agent platform and checked by this agent platform. His reasoning for the construct is that it might be easier to transfer this signature to the next agent platform (see step 2.15) without the need to transport the whole message.

<table>
<thead>
<tr>
<th>2.1</th>
<th>Get message from previous agent platform and check signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>If not true, COMPLAIN and STOP</td>
</tr>
<tr>
<td>2.3</td>
<td>Get state_{i+1}, sign_{i}(state_{i+1}) from message</td>
</tr>
<tr>
<td>2.4</td>
<td>Check sign_{i}(state_{i+1})</td>
</tr>
<tr>
<td>2.5</td>
<td>If not true, COMPLAIN and STOP</td>
</tr>
<tr>
<td>2.6</td>
<td>If (predecessor is untrusted) then</td>
</tr>
<tr>
<td>2.7</td>
<td>Get input_{i}, state_{i}, sign_{i-1}(state_{i}) from message</td>
</tr>
<tr>
<td>2.8</td>
<td>Check sign_{i-1}(state_{i})</td>
</tr>
<tr>
<td>2.9</td>
<td>If not true, COMPLAIN and STOP</td>
</tr>
<tr>
<td>2.10</td>
<td>Compute state_{i+2} from state_{i+1} using input_{i+1}</td>
</tr>
<tr>
<td>2.11</td>
<td>Check if state_{i+1} from line 2.10 and state_{i+1} received from predecessor differ</td>
</tr>
<tr>
<td>2.12</td>
<td>If true, COMPLAIN and STOP</td>
</tr>
<tr>
<td>2.13</td>
<td>Compute state_{i+2} from state_{i+1} using input_{i+1}</td>
</tr>
<tr>
<td>2.14</td>
<td>Transfer agent code to next agent platform if needed</td>
</tr>
<tr>
<td>2.15</td>
<td>Add sign_{i}(state_{i+1}), sign_{i+1}(state_{i+2}) to message</td>
</tr>
<tr>
<td>2.16</td>
<td>Add input_{i+1}, state_{i+1}, state_{i+2} to message</td>
</tr>
<tr>
<td>2.17</td>
<td>Sign message and send it to the next agent platform</td>
</tr>
</tbody>
</table>

**Figure 8.4:** Hohl’s optimised reference states protocol for untrusted agent platform i+1.

The checking phase encompasses steps 2.7 to 2.12 inclusive, but is only entered/performed if the previous agent platform is untrusted (step 2.6). This is because trusted agent platforms do not attack, and thus do not need to be checked for execution integrity. The agent platform first verifies that the signature of the agent platform before the previous agent platform on the initial state of the original computation on the previous state (state_{i}). After step 2.9 the current agent platform can be sure that the agent platform produced this state and the previous agent platform also gained this state by either checking the computation (in case that the agent platform before the previous agent platform is untrusted) or by knowledge that this agent platform is trusted. The inclusion of the signature sign_{i-1}(state_{i}) states this agreement and is enforced by the signature of the whole message by the previous agent platform. The current agent platform then checks the computation of the previous agent platform by re-executing the agent
using input_i and by comparing the resulting state with the received one. If the resulting states differ, the last agent platform attacked the agent. In this case, the agent user is notified and further processing of this protocol is stopped. The agent user has the evidence now to prove the modification was made (given the incriminating evidence is forwarded on by this agent platform who noticed the execution integrity breach), and to legitimately claim the previous agent platform attacked.

Then, the agent platform executes its normal computation, thus gaining the final state state_{i+2}. If needed, the agent code is transferred to the next agent platform. Finally, the current agent platform signs the final state and adds it to the message to be sent to the next agent platform. The signature of the previous agent platform signing the initial state of the computation was added, too, so the next agent platform can check the integrity of this state in order to check the computation of the current agent platform itself. The other necessary parts required for checking are also added to the message, it is signed and transferred to the next agent platform [85].

### 8.3.1.4 Trusted Agent Platform Protocol

Figure 8.5 lists Hohl’s reference states protocol steps for an arbitrary (in terms of the position in an agent’s itinerary) trusted agent platform.

Compared to the protocol for an untrusted agent platform (as described in Section 8.3.1.3), the trusted agent platform protocol only lacks the transfer of sign_i(state_{i+1}), input_{i+1}, and state_{i+1} to the next agent platform. These are the elements required for checking the previous agent platform’s execution of the agent. As this agent platform is trusted, they are therefore not needed at the next agent platform.

Finally, the protocol for the last agent platform needs only to check the previous computations by an untrusted agent platform. Therefore, it uses the protocol for a trusted agent platform (i.e. Figure 8.5) without the last four lines [85].

### 8.3.1.5 Reference States Limitations

Hohl’s reference states integrity execution approach is admirable particularly because:

- The execution integrity checks are performed *autonomously* in *real-time*, and
3.1 Get message from previous agent platform and check signature
3.2 If not true, COMPLAIN and STOP
3.3 Get state\(_{i+1}\), sign\(_i\)(state\(_{i+1}\)) from message
3.4 Check sign\(_i\)(state\(_{i+1}\))
3.5 If not true, COMPLAIN and STOP
3.6 If (predecessor is untrusted) then
3.7 Get input\(_i\), state\(_i\), sign\(_{i-1}\)(state\(_i\)) from message
3.8 Check sign\(_{i-1}\)(state\(_i\))
3.9 If not true, COMPLAIN and STOP
3.10 Compute state\(_{i+1}\) from state\(_i\) using input\(_i\)
3.11 Check if state\(_{i+1}\) from line 3.10 and state\(_{i+1}\) received from predecessor differ
3.12 If true, COMPLAIN and STOP
3.13 Compute state\(_{i+2}\) from state\(_{i+1}\) using input\(_{i+1}\)
3.14 Transfer agent code to next agent platform if needed
3.15 Add sign\(_{i+1}\)(state\(_{i+2}\)) to message
3.16 Add state\(_{i+2}\) to message
3.17 Sign message and send it to the next agent platform

**Figure 8.5:** Hohl’s optimised reference states protocol for trusted agent platform \(i+1\).

- The scheme aims to non-refutably tie stakeholder inputs with a mobile agent’s execution sessions.

However, we see a number of limitations with the scheme. These limitations include (listed in no particular order of weighting):

1. Agent platform collaboration attacks are possible, particularly two or more consecutive agent platforms in an agent’s itinerary maliciously corroborating to cover breaches of execution integrity (including malevolent agent platform inputs and improper interpretation of the agent’s code). For example, considering the quality research paper/author finder mobile agent application description in Section 8.1.1, two consecutive malicious agent platforms could collaborate to hide a false input or calculation by the first malicious collaborating agent platform, thus misconstruing the agent’s top \(n\) research paper/author mission results which are returned to the agent user.

2. Agent platform inputs cannot be kept secret from checking agent platforms, due to the protocol’s requirement that these inputs be forwarded for execu-
tion checking on the next agent platform. However, agent platforms in the agent’s itinerary may be competitors, or at least not trusting the other’s sincerity or practices. For example, once again considering the quality research paper/author finder mobile agent application description in Section 8.1.1, one agent platform recommending principal may well not want the next (or subsequent) agent platform recommending principal(s) to know his top \( n \) research paper/author and associated rating recommendations.

3. There is no robustness in terms of preventing denial-of-service attacks from agent platforms, who may refuse to service agents or not abide by the reference states protocol.

4. There is no possibility for agent user or itinerary anonymity, since stakeholders must digitally sign their inputs according to the protocol: The agent user signs its agent; agent platforms, on behalf of their owners, sign resulting calculated states for agents and their inputs are encapsulated in signed messages. For example, once again considering the quality research paper/author finder mobile agent application description in Section 8.1.1, the agent user may not want its agent’s itinerary disclosed to agent platforms - thus, disclosing its potentially sensitive business associates which may conflict with existing business associates.

5. Furthermore to the previous limitation, the scheme necessitates that agent platforms know the agent user trust statements in agent platforms. Both may be viewed as breaches of privacy, and could result in mining agent user preferences and an aggressive malicious campaign by agent platforms against preferred competitors.

6. The protocol is overly complex, for example requiring agent platforms to sign the resulting state element and then sign the encapsulating whole message as well. In addition, it is not clear why the agent code is not transferred with the signed message - how is it protected, and how does an agent platform link agent code to its pertinent reference state message?

7. There is only one mode for the reference states protocol - perform the execution checks as a pre-condition that must be met before executing the agent on the agent platform. This will increase an agent platform’s processing load significantly (i.e. double it on average), and the agent user may
Chapter 8. Protecting the Execution Integrity of Mobile Agents

- due to mission urgency reasons - not always wish for the agent’s execution checks to be performed until after the agent has travelled to all agent platforms.

8. The robustness of long-term stakeholder non-repudiation is questionable. For example, how are the reference states maintained? Does each agent platform append to one long, growing reference state message list that is returned to the agent user along with the agent at the end of the agent’s mission? If any agent platform in the agent’s itinerary destroys the agent reference state, previous non-refutable agent platform statements are broken unless each agent platform maintains a log of all its transported reference state messages. Checking would then be instigated spasmodically by an agent user, triggered only on their suspicion that something might have gone wrong - breaking the definition of autonomous, asynchronous execution for mobile agents.

8.3.1.6 Summary of Hohl’s Reference States Scheme

In Section 8.3.1 we provided an in-depth discussion of Hohl’s reference states execution integrity strategy for mobile agents, including an explanation of his optimised reference states protocols that agent platforms must follow in conforming with the scheme. Whilst we pointed out that the objectives of Hohl’s reference states scheme are admirable, we listed a number of limitations with the reference states protocols in Section 8.3.1.5. Nevertheless, the scheme can be modified appropriately to work more safely within the MASHIn framework. This is exactly what we do in the next section, with the modified reference states scheme forming an important part of our MASHIn RECDAM execution integrity approach.

8.3.2 Reference States Adaptations to MASHIn

In this section we will discuss the necessary agent itinerary changes which are critical in working towards a more secure reference states model, concluding with our RECDAM adapted reference states execution integrity protocols for the MASHIn in Section 8.3.2.3. We discuss three modes in which our RECDAM execution integrity approach can conceivably operate in Section 8.3.3, and in Section 8.3.4 we discuss how the feedback from execution integrity checks in the RECDAM scheme can be flexibly utilised to respond to problematic events (such as a discovered
breach of execution integrity) and play an important role in discouraging future attacks.

8.3.2.1 General Case Incorporating Checking Hosts

We will now explain our adaptation of Hohl’s reference states checking mechanism and its subsequent incorporation in the MASHIn RECDAM framework. Referring to the example agent itinerary scenario depicted in Figure 8.6, an agent’s itinerary consists of three remote agent platforms (labelled 1, 2, 3 for successive visitation). The evenly dotted arrowhead lines represent a possible itinerary taken by an agent if Hohl’s original reference states protocol was applied in the MASHIn context - with the provision that the MASH could be considered the agent’s home agent platform. The solid arrowhead lines represent the corresponding itinerary under our adapted reference states scheme. The unevenly dotted and solid lines are the forwarding of a copy of the agent (including ‘real’ callback classes) for mobile agent privacy, as discussed in Section 7.3.2.

The checking hosts are MASH-delegated agent platforms inserted in the agent’s itinerary between remote agent platforms. Strategic roles for checking hosts in the MASHIn were first touched on in Section 6.5, when it was pointed out that they are vital in helping track agent location and improving the chances of an agent’s mission completion through fault tolerant handling. In terms of execution integrity for mobile agents in the MASHIn, the checking hosts play a vital purpose by verifying claimed reference states from untrusted agent platforms. In the general case, they are the only entities in the MASHIn (besides the MASH once the agent returns there), to perform execution checks - which contrasts markedly from Hohl’s reference states strategy where execution checking for an agent platform is made immediately on the next agent platform [85].

With the introduction of the trusted checking hosts into the adapted reference state model, the two major weaknesses acknowledged by Hohl in his scheme [85] (and discussed in the reference states limitations list that we constructed in Section 8.3.1.5) are eliminated. Collaboration attacks are now overcome because there are now no two untrusted agent platforms in succession - due to the indirection via checking hosts. Moreover, the inputs from one agent platform are kept private from the next logical agent platform in the agent’s itinerary as only the checking host sees the input, and it has no stakeholder application interest in the agent’s data or the previous agent platform’s input [70].
Figure 8.6: Adapted reference states agent movement for use in MASHIn RECDAM scheme.
Interestingly, to overcome the first abovementioned weakness in Hohl’s reference states scheme (i.e. the very real possibility of agent platform collaboration attacks), Hohl suggests an alternative protocol which introduces \( n+1 \) additional agent platforms per mission to tolerate \( n \) malicious agent platforms [85]. Essentially we have done this in the MASHIn RECDAM approach, except for two significant differences:

- As can be seen in detail in Section 8.3.2.3 the adapted reference states protocols for the MASHIn are significantly simpler (e.g. no assumptions about agent platform trust worthiness and public disclosing of this information) and more efficient (e.g. no need to sign both the resulting agent state and the whole message) than Hohl’s protocols.

- The inserted checking hosts play pivotal roles not only in reliably detecting breaches in the execution integrity of agents processed at remote agent platforms, but also in adding fault tolerance capabilities to an agent’s travels as mentioned. Furthermore, they - along with the MASH - enable agent user and agent itinerary anonymity (see Section 6.5), and also eliminate the second major problem identified by Hohl in his scheme (i.e. the impossibility of keeping agent platform inputs private from each other).

Hence, with the strategic incorporation of MASHIn TTP MASHs and checking hosts along with our revised set of reference states protocols (see Section 8.3.2.3), most if not all of the limitations identified with Hohl’s reference states scheme in Section 8.3.1.5 have been overcome, or can be feasibly overcome, in the context of the MASHIn RECDAM scheme.

8.3.2.2 Flexibility to Skip Checking Hosts

One may initially think the general MASHIn RECDAM case of an agent travelling to a checking host (in between every agent platform in its itinerary) to perform execution integrity checking is not a very efficient use of network resources. However, the security benefits of this design basically mandate this as the default behaviour for safe real-time in-mission execution integrity checking. This design for real-time in-mission execution integrity checks, by hopping to a TTP after every execution session at an agent platform, is mandated in the adaptation of Vigna’s execution tracing mechanism [212], as proposed by Tan and Moreau in [194] and discussed in Section 8.2.3. Whilst their scheme is a big
improvement on Vigna’s original execution tracing mechanism, it is significantly more vulnerable to collaboration attacks between verification servers (analogous to MASHIn RECDAM checking hosts) and agent platforms because the agent platforms associate themselves with, and utilise, a verification server for agent execution integrity checking. In the MASHIn, however, the MASH (who chooses appropriate checking hosts for an agent’s mission) is a genuine TTP neutral towards both major stakeholders, as are the delegate checking hosts. Moreover, neither major stakeholder (i.e. agent user nor agent platform owner) chooses the checking hosts to be utilised in the MASHIn RECDAM scheme.

It would be possible to extend our RECDAM protocols (listed in Section 8.3.2.3) to incorporate the flexible option to not always go to a checking host after every agent platform, for example in the case that the agent user trusts consecutive agent platforms in its itinerary. However, one has to be very careful here because for this flexibility to be realistic the protocol designer should also (to be fair to both stakeholders) ensure that the two or more consecutive agent platforms trust each other in terms of performing correct (i.e. non-attacking and correct) execution of agents, and the agent platforms must also trust the agent user (to not raise later disputes). However, these trust relationships might not be always bidirectionally identical, or even trivial to ascertain. For all of these above reasons, we strongly recommend staying with the general case RECDAM model (i.e. hopping to a checking host between agent platforms to process execution integrity checks).

8.3.2.3 Adapted Protocols

We have formulated a revised set of reference states protocols for use in the MASHIn RECDAM scheme. As pointed out in Section 8.3.2.1, in the general execution integrity processing case in the MASHIn, only checking hosts perform reference state checks - with this adaptation assisting in the addition of significant security benefits over the original reference states scheme. In terms of protocol overheads, the changes are not as costly as one would expect.

We must point out that all messages sent in the following protocols are also protected for confidentiality using asymmetric encryption, by enveloping the messages for the intended recipient. This step was also not explicitly included in Hohl’s original reference states protocols (discussed in Sections 8.3.1.2, 8.3.1.3 and 8.3.1.4) as this is standard, accepted safe practice when considering proto-
protocols related to the migration of agents [19]. Where possible, in explaining our adapted new protocols, we try to remain consistent with the notation employed in our explanation of Hohl’s protocols (see Section 8.3.1), specifically to identify agent platforms (i.e. agent platform $i+1$) and resulting agent states from executing at those respective agent platforms (i.e. state $i+2$).

First we consider the RECDAM protocol for the MASH that was delegated mission protection of the mobile agent, as listed in Figure 8.7. The MASH is responsible for alerting the first checking host that it is to expect a mobile agent for reference states processing. In the protocol messages between MASHs and checking hosts, we have not included information such as the identity of the agent to expect and from which agent platform the agent will arrive from because we do not want to complicate explanation of the RECDAM protocols; we are primarily interested in comparing the adapted reference state checking functionality in our RECDAM scheme with Hohl’s original scheme\(^5\). Such information can be trivially incorporated into the following protocols if the MASHIn were to be constructed, thus necessitating a more detailed design.

Considering Figure 8.7, the MASH computes and adds the initial agent state to messages being constructed for both the first agent platform and first checking host in the agent’s itinerary. It is important to note a semantic difference here: the first agent platform in Hohl’s scheme (i.e. “Agent Platform 1”) is the agent’s home agent platform, whereas in our scheme this is instead synonymous to the MASH; the agent platform designated as “Agent Platform 1” in Figure 8.6 depicting our scheme is the first potentially untrusted agent platform which would correlate to Agent Platform 2 in Hohl’s scheme. We chose to make this distinction because the agent’s execution integrity at Agent Platform 1 (in our scheme) is checked at Checking Host 1; this slight difference in semantics assists in communicating our upcoming interdependent RECDAM agent platform and RECDAM checking host protocols.

Contrary to Hohl’s original scheme, we include the agent code in the built up messages for the next agent platform and checking host. This alleviates the need for the receiver to wait for and associate two messages (i.e. the agent’s code and reference states messages). In the message for the first checking host, the code message element also includes the ‘real’ callback classes spoken about in

\(^5\)Hohl, in [85], has not included all necessary information needed for correct interpretation of his protocols either - for example, how do agent platforms know whether the previous agent platform was trusted or untrusted?
Section 7.3.2 for achieving mobile agent data and code privacy. Considering the quality research paper/author finder mobile agent application example discussed in Section 8.1.1, the agent user’s competency ratings in the agent platform recommending principals would be one possible data privacy example. The option to keep searching recursively (using the recommender principals’ recommended principals) is one possible code privacy example.

The two messages are then digitally signed by the MASH and forwarded to their respective intended recipient.

<table>
<thead>
<tr>
<th>MASH.1</th>
<th>Compute state$_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASH.2</td>
<td>Add state$_2$ to message for first checking host</td>
</tr>
<tr>
<td>MASH.3</td>
<td>Add state$_2$ to message for first agent platform</td>
</tr>
<tr>
<td>MASH.4</td>
<td>Add agent code (including ‘real’ callback classes) to message for first checking host</td>
</tr>
<tr>
<td>MASH.5</td>
<td>Add agent code (not including ‘real’ callback classes) to message for first agent platform</td>
</tr>
<tr>
<td>MASH.6</td>
<td>Sign message intended for first checking host and send it</td>
</tr>
<tr>
<td>MASH.7</td>
<td>Sign message intended for first agent platform and send it</td>
</tr>
</tbody>
</table>

**Figure 8.7:** RECDAM protocol for the MASH (i.e. an agent’s “home” agent platform in the MASHIn).

We now consider, in Figure 8.8, the protocol for an arbitrary agent platform in our RECDAM execution integrity scheme. In contrast to Hohl’s reference states protocols, we do not have a separate protocol for trusted and untrusted agent platforms. This removes the need for an agent platform to need to know if it is trusted by the agent user (as necessitated in Hohl’s trusted versus untrusted protocols). Moreover, it provides the means by which the computation and inputs from the agent platform leading to a resulting state for an agent can be non-refutably tied for execution integrity to the agent platform. This is important because the agent platform may not trust the agent user to dispute the execution integrity of the agent platform at a later time (i.e. the trust relationship may not be bi-directional between agent user and agent platform). This is another example of whereby the MASHIn approach is admirable, by taking the interests of both major stakeholders into consideration, facilitating the probability of bringing mistrusting major stakeholders together into a safe working relationship.

The agent platform gets the message from the previous checking host in the
8.3. Real-time Execution Checking and Deterrent Against Misbehaviour

AP.1 Get message from previous checking host and check signature
AP.2 If not true, COMPLAIN and STOP
AP.3 Get state\(_{i+1}\) from message
AP.4 Compute state\(_{i+2}\) from state\(_{i+1}\) using input\(_{i+1}\)
AP.5 Add agent code to message
AP.6 Add input\(_{i+1}\), state\(_{i+2}\) to message
AP.7 Sign message and send it to the next checking host

| Figure 8.8: RECDAM protocol for agent platforms. |

agent’s itinerary, and checks the digital signature. If the signature cannot be verified, the sender checking host is informed of this problem. If a verifiable message is not sent in-return to the agent platform, the agent platform does not continue with processing the protocol. In step AP.3, the agent platform gets the agent’s initial state state\(_{i+1}\) from the message, and computes the resulting state state\(_{i+2}\) from executing the agent using input input\(_{i+1}\) from its agent platform environment in step AP.4. The input and final agent states from processing the agent at this agent platform are then included in the applicable element positions of the message to be sent to the next checking host (step AP.6). This is done, as explained above, regardless of whether the agent user trusts the agent platform - for execution integrity non-repudiation from the agent platform’s perspective.

The message is digitally signed by the agent platform and securely sent to the next intended checking host in the agent’s itinerary.

This RECDAM agent platform protocol is considerably more efficient than either of Hohl’s untrusted or trusted reference state protocols (discussed in Sections 8.3.1.3 and 8.3.1.4 respectively). This is because steps 6-12 inclusive in Hohl’s protocols are not required in the RECDAM protocols, since agent platforms no longer perform the reference states checking functionality - not only ensuring the execution integrity process is made considerably more secure, but also facilitating the agent platform in providing better (i.e. more efficient and reliable) servicing of mobile agents.

We now consider, in Figure 8.9, the protocol for an arbitrary checking host in our RECDAM execution integrity scheme. In some respects it is quite similar to the agent platform protocols in Hohl’s original reference states scheme, because it is the checking hosts in the MASHIn RECDAM scheme which perform the actual
execution integrity checks (not agent platforms).

The checking host receives two messages - the first from the previous checking host, or the MASH if it is the first checking host in the agent’s itinerary; and the message from the previous agent platform for which it is performing the execution integrity session check. Steps CH.1-CH.4 inclusive are receiving these two messages and verifying that they are not broken via the digital signatures of the two sending entities. As in the previous protocols, if the message signatures can not be verified, a request is sent to the pertinent sender/s to re-send a correctly signed message. If such a message is not received, processing of the protocol does not proceed.

In step CH.5, the claimed resulting agent state from the agent’s execution on the previous agent platform is extracted. This above state extraction is needed here, in contrary to Hohl’s original scheme [85], regardless of if the previous agent platform is untrusted because the MASHIn RECDAM scheme protects non-repudiation of execution integrity from the agent platform (owner) stakeholder perspective as well. This is why the conditional step 6: “If (previous agent platform is untrusted)” in Hohl’s scheme is not in our protocol here. Remember, as we noted above, even though the agent user might trust an agent platform the trust relationship might not be asymmetrical (i.e. the agent platform owner may not trust the agent user).

Steps CH.6-CH.10 are the actual execution checking steps. Firstly, the checking host must extract the claimed inputs from the previous agent platform’s execution session with the agent. The initial state of the agent on the previous agent platform is then extracted from the previous checking host’s message (step CH.7). This makes it more secure (against agent platform collaboration attacks, for example) then the case in Hohl’s original scheme whereby only agent platforms were making these initial agent state claims. Step CH.8 is the re-execution of the agent execution session on the previous agent platform. If the re-executed resulting state differs from the previous agent platform’s purported resulting state the discrepancy is noted to the appropriate authorities (this would be the MASH in the first instance), and the protocol processing is ceased.

If however the execution integrity check was verified, appropriate messages are constructed and sent to the agent’s next checking host and agent platform in steps CH.11-CH.16 inclusive (these steps are logically similar to the last six steps in the RECDAM MASH protocol in Figure 8.7). Note that these six last steps
8.3. Real-time Execution Checking and Deterrent Against Misbehaviour

CH.1 Get message from previous checking host or the MASH, and check signature
CH.2 If not true, COMPLAIN and STOP
CH.3 Get message from previous agent platform and check signature
CH.4 If not true, COMPLAIN and STOP
CH.5 Get state\textsubscript{\textit{i}+1} from previous agent platform message
CH.6 Get input\textsubscript{\textit{i}} from previous agent platform message
CH.7 Get state\textsubscript{\textit{i}} from previous checking host/MASH message
CH.8 Compute state\textsubscript{\textit{i}+1} from state\textsubscript{\textit{i}} using input\textsubscript{\textit{i}}
CH.9 Check if state\textsubscript{\textit{i}+1} from line CH.8 and state\textsubscript{\textit{i}+1} received from previous agent platform differ
CH.10 If true, COMPLAIN and STOP
CH.11 Add state\textsubscript{\textit{i}+1} to message for next checking host
CH.12 Add state\textsubscript{\textit{i}+1} to message for next agent platform
CH.13 Add agent code (including ‘real’ callback classes) to message for next checking host
CH.14 Add agent code (\textit{not} including ‘real’ callback classes) to message for next agent platform
CH.15 Sign message intended for next checking host and send it
CH.16 Sign message intended for next agent platform and send it

\textbf{Figure 8.9:} RECDAM protocol for checking hosts.
are not processed if this “checking host” is the MASH (i.e. the agent is back at the MASH for the final time after visiting the last agent platform in its itinerary, and the MASH is simply acting in the role of a checking host at this final point in the agent’s itinerary). There is one caveat to be aware of in step CH.11 and CH.12. The agent state to be transported to the next checking host and agent platform is in fact the agent state after the secure ‘real’ callback class/es (privy only to the MASH and checking hosts) have been applied; the resulting agent state may or may not be the same. The ‘real’ agent state component in the ‘real’ callback classes is forwarded to the next checking host only (i.e. not the next agent platform) - encapsulated in step CH.13.

Even though in our RECDAM scheme the checking host is receiving two messages (i.e. one from the previous checking host and one from the previous agent platform) the actual protocol for the execution checking functionality is near even in terms of efficiency of secure protocol steps to Hohl’s reference states scheme [85, 86]. The big difference in the schemes, as we have highlighted, is that our execution integrity scheme is considerably more secure and robust for both major stakeholders.

8.3.3 Modes of Operations

Our MASHIn RECDAM scheme for mobile agent execution integrity, discussed in detail above (in Section 8.3.2), could conceivably be implemented to work in at least three different modes of operations. We will briefly consider each possibility in turn from Section 8.3.3.1 through to Section 8.3.3.3, and in Section 8.3.3.4 we discuss a possible fee structuring model to support execution integrity checking.

8.3.3.1 Real-time In-Mission Checking

Real-time in-mission checking is the most secure method/mode feasible for maintaining the execution integrity of mobile agents, without using tamper-proof trusted environments (which we have previously ruled out, in Section 5.1.3, as being impractical for widespread Internet mobile agent use).

This mode of operation involves execution integrity checking of a mobile agent’s sessional work at an agent platform directly after it has finished execution at that agent platform (as described in Section 8.3.2), with supplied protocols in Section 8.3.2.3. The execution integrity checking is done on checking hosts before the agent’s secure ‘real’ callback classes are processed.
8.3. Real-time Execution Checking and Deterrent Against Misbehaviour

8.3.3.2 Background In-Mission Checking

In this mode, the checks are performed sometime after the agent’s secure ‘real’
callback classes (if any) are processed on the checking host and the agent has
been forwarded onto the next agent platform in the agent’s itinerary.

If any discrepancies in execution integrity are subsequently uncovered the
checking host will notify the MASH delegated mission protection of the agent’s
mission, who can take appropriate action.

This execution integrity mode can be employed for less critical agent missions.

8.3.3.3 Post-Mission Checking

In this mode, the MASH can coordinate the checking hosts to unconditionally
re-execute the agent’s mission from its itinerary start to end points using the ir-
refutable agent platform sessonal inputs and agent states collected in the original
mission run. This mode of operation is similar to some existing execution tracing
schemes (such as execution tracing [212]) except those schemes are only triggered
on a suspicion of the agent’s user, thus they contradict the desired properties of
safe and autonomous mobile agents.

Both the post-mission and background in-mission execution checking modes
are acting, minimally, as a real-time deterrent router against potentially malicious
agent platforms (who remain unaware of what execution integrity checking mode
is being employed).

8.3.3.4 Fee-Charge Structure

All three aforementioned RECDAM modes facilitate long-term non-repudiation
of execution integrity for both major stakeholders. Regardless of which mode is
utilised in a mobile agent mission scenario, both major stakeholders should be
charged for utilising the MASHIn RECDAM service, as it in both their interests
that mobile agents are checked for execution integrity and the MASHIn TTPs
need to be remunerated for providing such a service.

MASHIs can coordinate, via checking hosts, a billing charge per execution
integrity checking transaction. Utilisation of the real-time in-mission mode of
RECDAM operation should probably be charged at a higher fee than the other
two modes because it is generally-speaking a more resource-intensive mode of
processing for checking hosts (i.e. the checking hosts have less flexibility to load
balance the processing of execution checks). Continuing this line of reasoning, utilisation of the post-mission checking mode should probably be charged at a higher fee than utilisation of the background in-mission checking mode.

The major stakeholders (both agent users and agent platform owners) should also be charged for this service based on their inputs into the agent’s execution sessions (i.e. agent code processing overhead, and agent platform input count/size).

Lastly, the major stakeholders should also be charged proportionally to the period length of time (e.g. number of minutes/hours/months/years) they wish the checking hosts to retain the irrefutable RECDAM execution state messages for. This helps to better facilitate the management and usage of checking host secondary storage media.

The Transaction Non-repudiation Module (discussed in Section 6.1.3) is responsible for initiating stakeholder billing for MASHIn TTP security services delivered.

### 8.3.4 Feedback Handling

The TTP event-monitoring capabilities in the MASHIn RECDAM process promote a more robust environment for autonomous mobile agents to work in. In this section, we will briefly provide an overview of two events which can have event-response-handlers from both an agent user and infrastructure perspective. The first event we consider is that an agent’s TTL for processing at an agent platform (in-between being transported from/to checking hosts) could expire. Secondly, an execution integrity breach by an agent platform could be detected.

#### 8.3.4.1 Unresponsive Agent Platform Exceeds Agent TTL

In Section 6.5 we discussed how the MASH can coordinate the checking hosts in an agent’s itinerary to monitor the agent for expiry of its TTLs. The TTL for an agent travel between checking hosts includes the time taken to the agent platform in-between checking hosts and the execution processing time at that agent platform.

Regardless of if there is TTL expiration for an agent, the monitoring checking host (i.e. in the agent’s itinerary after the agent platform to which the agent was sent) has been sent a copy of the agent’s code/state including secure ‘real’ callback classes by the previous checking host. We spoke about the minimal
8.3. Real-time Execution Checking and Deterrent Against Misbehaviour

base methods in these callback classes in Section 7.3.2. More agent user event-response-handling methods can be envisaged; in this case a method to respond to TTL expiration. The agent user could embed code in this method to respond to this TTL expiration event however it chooses, including bypassing the non-responsive agent platform hop and going onto the next agent platform in its itinerary. Additionally, the agent user could wish to revise the agent’s mission itinerary to retry that agent platform at a later time - for example when it is back at the MASH.

From an infrastructural perspective there can be MASHIn MASH event-response-handlers including stationary agents at MASHs (coordinating with checking hosts) which monitor agent platform responsiveness. If a number of agent’s time-out at, or in-transit to/from, an agent platform a MASH could send one or more of its own agents to the agent platform to try to diagnose the problem better. Subsequently it could diagnose whether the agent platform is responding slowly under a heavy processing load, or whether the network connection to/from the agent platform is the cause of the repeated delays in agents passing to/from that agent platform.

8.3.4.2 Agent Handler for Execution Integrity Breach

The MASHIn RECDAM process can detect breaches of execution integrity by an agent platform as discussed in Section 8.3.2.3. For example, considering the quality research paper/author mobile agent application description in Section 8.1.1, if a breach of execution integrity by a recommending agent platform is detected by a checking host, then this feedback can be appropriately incorporated in the agent user’s future business dealings with the offending agent platform.

Once again, a copy of the agent’s valid code/state (before travelling to the breaching agent platform) has been forwarded to the next checking host. Similar to the TTL expiry agent user event-handler, a secure ‘real’ callback method can be defined to include actions to take on the event on a detected agent platform execution integrity. For example, the agent could be programmed to continue on (using its pre-agent-platform-forwarded integral code/state) to complete the remainder of its itinerary, or it could like to retry executing at the agent platform again to see if the attack was random or a repeating behaviour. In addition, or as an alternative, it could choose to stop processing its remaining itinerary and/or securely message (from its safe haven at the checking host) any cooperating mis-
sion agents or agent user stakeholder associates to notify that the offending agent platform should be treated with extra caution.

8.3.4.3 Feedback as Raw Data Input into Reputation Calculations

The output from these two aforementioned (i.e. agent TTL and execution integrity checking) and other conceivable RECDAM monitoring processes such as mobile agent virus and/or erratic code logic scanning could be used to formulate stakeholder reputation/confidence scores in other stakeholders. Subscribing stakeholders could submit property weightings to a MASH who would transparently calculate private, personalised stakeholder reputation/confidence scores based on the stakeholder-defined weightings and the raw feedback from MASHIn TTP event-monitoring. Dynamic updating of these calculated scores would be possible as new data is fed in from reliable first-hand observations.

Discussion on these strategic possibilities, having the aim to bring unknown and/or mistrusting (but well-performing and non-malicious) mobile agent stakeholders together, for the MASHIn are discussed in Chapter 9.

8.4 RECDAM Comparison with Existing Strategies

Table 8.2 compares the capabilities of existing execution integrity strategies for mobile agents with our RECDAM strategy capabilities. Our RECDAM strategy satisfies all of the desired execution integrity objectives (listed and discussed in Section 8.1) for a robust execution integrity strategy, whilst the existing strategies did not fare nearly as well\(^6\) - notably, none of them bettering support for more than four of the seven desired ("present"/"non-present") properties:

- Application-Independent (AI)
- Dynamic Itinerary Possible (DIP)
- Real-Time Checking (RTC)
- Unconditional Post-Mission Autonomous Checking (UPMAC)
- Stakeholder Denial-of-Service Resistant (SDoSR)

\(^6\)The existing execution integrity strategies were reviewed and critiqued earlier in Section 8.2.
• Agent Platform Collaboration Attack Resistant (APCAR)

• Real Punishment Deterrent (RPD)
<table>
<thead>
<tr>
<th>Scheme</th>
<th>AI</th>
<th>DIP</th>
<th>RTC</th>
<th>UPMAC</th>
<th>SDoSR</th>
<th>APCAR</th>
<th>RPD</th>
<th>Effectiveness</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial Result Encapsulation (PRE)</td>
<td>√</td>
<td>%</td>
<td>%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Medium; often relies on agent user/home platform to perform checks</td>
<td>Medium-Good</td>
</tr>
<tr>
<td>Holographic Proofs (HP)</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Impractical now, and likely to remain so</td>
<td>Poor</td>
</tr>
<tr>
<td>Execution Tracing (ET)</td>
<td>√</td>
<td>√</td>
<td>+</td>
<td></td>
<td></td>
<td>+</td>
<td></td>
<td>Medium; but the cryptographic trace management is problematic</td>
<td>Poor-Medium</td>
</tr>
<tr>
<td>Reference States (RS)</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Poor-Medium, because of the collaboration attack and input privacy problems</td>
<td>Medium-Good</td>
</tr>
<tr>
<td>RECDAM</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>Very high, more secure and reliable than reference states</td>
<td>Low-Medium, but very bearable given the high protection afforded by the scheme</td>
</tr>
</tbody>
</table>

Table 8.2: A comparison of existing mechanisms for mobile agent execution integrity with our RECDAM scheme.

Table Symbol Key: “√” = feature supported; “+” additional supported feature in the extended execution tracing method; “%” = supported in some forms of scheme.
In analysing those properties with respect to the RECDAM strategy we see:

- The RECDAM strategy is an execution integrity approach supporting arbitrary mobile agents, meaning it can support *application-independent* mobile agents and those agents are not limited by execution code size.

- In the RECDAM scheme, mobile agent *dynamic itinerary changes* are facilitated via mobile agent secure ‘real’ callback classes (specifically the event-response methods encapsulated therein) being retained on trusted checking hosts as discussed in Section 8.3.4. Travel to additional agent platforms may also be negotiated autonomously when the agent returns to the MASH; importantly the agent execution sessions on these additional agent platforms are also protected under the RECDAM process because the MASH is capable of coordinating checking hosts for protecting this additional travel and agent execution.

- *Real-time in-mission agent checking* is supported on the checking hosts, via an adaptation of Hohl’s reference states scheme. However, as pointed out in Section 8.3.2, the RECDAM scheme is significantly more secure than Hohl’s original scheme.

- In Section 8.3.3, we discussed three modes that could conceivably be supported for RECDAM execution checking of mobile agents. One of those modes was *unconditional autonomous checking post-mission*, but from a short and long-term security perspective the real-time in-mission mode for which we supplied proposed protocols for in Section 8.3.2.3 is the preferred mode.

- The RECDAM approach is both *resistant against major stakeholder denial-of-service* and *resistant against agent platform collaboration attacks* as discussed in Section 8.3.2.

- Finally, as we hinted on in Section 8.3.4.3 and we will expand on in Chapter 9, the MASH in RECDAM offers a *real punishment deterrent* for breaches of execution integrity by the possibility of lowered stakeholder reputation/confidence scores against an offending agent platform.

To a degree, there is an increased cost in the MASH in RECDAM scheme in terms of *efficiency* because agents, and their execution integrity specifically, are
checked on checking hosts. This transport cost (agent travel to/from a checking host) is very bearable, though, considering the increased security benefits over existing schemes. One prominent example of this increased security robustness is the major improvements seen in the RECDAM scheme over Hohl’s reference states scheme, which we appropriately adapted and incorporated in our scheme.

Another drawback or limitation of RECDAM (and real-time in-mission execution checking schemes in general) is the potential for one major stakeholder to want a different agent itinerary pattern than the other major stakeholder. Consider the scenario whereby an agent user, for reasons of efficiency and/or trust perhaps, does not want the execution checks to be performed real-time in-mission - but, post-mission and still autonomous. However, an intermediate agent platform in the agent’s itinerary may want real-time in-mission checks, because it does not trust other agent platforms to interact with the agent ethically before or after visit to its agent platform. In this scenario, how do the stakeholders proceed to interact without deviating from their desired agent itinerary pattern? Some further negotiation or compromise needs to be made in such scenarios. In Section 9.5 we discuss some generic MASHIn limitations in terms of moving towards a production system that need future examining.

Provided the infrastructural TTP components are present, our RECDAM scheme is very effective and highly admirable since it does not favour protection of one stakeholder over the other. Security for both major mobile agent stakeholders is catered for in the MASHIn, which works to lower the barriers of mistrust so that they can more safely form and participate in working relationships.

8.5 Summary

In the previous chapter we demonstrated a new approach to achieving mobile agent privacy. Our approach, the Secure Callback Classes for Mobile Agent Privacy (SCC4MAP) scheme, utilised MASHIn TTPs (particularly MASHs and checking hosts) to deliver a flexible and very effective new form of mobile agent privacy.

In this chapter we concentrated on mobile agent execution integrity - at least as important a security property as mobile agent privacy for adding much

---

7A prior agent platform may try to insert a virus or logic bomb attack into the agent’s payload; a following agent platform may try to manipulate non-repudiation data from previous agent platform/s.
needed credibility to the safety of the Internet mobile agent paradigm. Like the SCC4MAP scheme, and the MASHIn design in general, our work on formulating a secure execution integrity strategy is not biased towards protecting one or the other mobile agent paradigm major stakeholders. Our strategic incorporation of genuine TTP entities is pivotal in continuing this macro-level approach, helping to lower the mutual mistrust lingering between the major stakeholders.

In Section 8.1 we began the chapter by defining a number of desired features for a robust execution integrity scheme. We also provided another mobile agent application example to highlight the need for a robust execution integrity scheme to be capable of protecting mobile agents which are not confined to application-specifics (e.g. only dealing with “window-shopping”), static-itinerary, limited autonomy, and so on.

We provided a critical overview of the major existing execution integrity strategies for mobile agents in Section 8.2, including discussion on their positive aspects and limitations for widespread autonomous mobile agent utilisation. None of the existing schemes came close to offering the desired capabilities of a robust execution integrity scheme. However, we found Hohl’s reference states [85, 86] scheme the most admirable because its execution checks are performed in real-time periodically during the course of an agent’s mission, it is not limited to specific application mobile agents, and it does not require the execution checks to be performed by the agent user or at the agent home platform.

Thus, we chose to build and improve on Hohl’s reference states scheme. Our first objective in Section 8.3 was to discuss the weaknesses of the reference states strategy; in doing so we also took a look at the reference states protocols. We then revised the protocols and demonstrated how execution integrity can be more robustly supported within the MASHIn framework. We also provided a guide on how MASHIn TTP entities can be remunerated for their execution integrity services, feasible modes of operation for execution integrity checking in the MASHIn, and how appropriate responses (flexibly from an agent user and infrastructure perspective) to execution integrity breaches can be formulated within the MASHIn framework.

In Section 8.4 we compared the existing mobile agent execution integrity strategies with our MASHIn Real-time Execution Checking and Deterrent Against Misbehaviour (RECDAM) scheme. From Table 8.1 on page 170, one can see our scheme fared favourably in general, and was the only strategy which provided the
capabilities of unconditional post-mission autonomous execution integrity checking, strong resistance against agent platform collaboration attacks, and provided a genuine capability to be a real-deterrent against breaches of execution integrity to both major stakeholders.

Our RECDAM scheme relies on an infrastructure of MASHs and checking hosts, both genuine TTPs, to perform execution integrity checks. In some regard our checking hosts are analogous to Tan and Moreau’s verification servers [194] in their scheme which improves on Vigna’s original execution tracing scheme [212]. The big difference between our scheme and Tan and Moreau’s, besides the fact we improve on Hohl’s reference states scheme and they improve on Vigna’s execution tracing mechanism, is the fact that our checking hosts are genuine TTPs (chosen by a non-biased MASH). In the improved execution tracing mechanism by Tan and Moreau, the agent platform chooses the verification server to perform execution integrity checks, providing a possible conflict of interest and the real concern of collaboration attacks between the verification servers and agent platforms.

Because our TTPs are non-biased towards either of the major mobile agent paradigm stakeholders, they are in an excellent position to provide reliable first-hand observations on fair and malicious stakeholder behaviour. We will discuss the strategic advantages for the MASHIn that this capability provides in the next chapter (i.e. Chapter 9). Not only is there a strong deterrent against misbehaviour (because attacks are more easily detected and attributed to the correct source in the MASHIn), but previously mistrusting but well-behaved mobile agent paradigm major stakeholders (identified from a reliable, unbiased track record) can be bought together for the initiation of useful work.

In Chapter 10 we discuss some network design considerations, and take a brief look at promising new research directions, that will be useful in helping protect MASHs and checking hosts - the MASHIn TTP entities on which much safeguarding of stakeholder security (and thus business-continuity) interests is dependent.
Chapter 9

A Framework for Transparent Confidence Scores in Uniting the Major Stakeholders

A good name, like good will, is got by many actions and lost by one.

Lord Jeffery

In the previous chapter we detailed a new, improved execution integrity strategy for Internet mobile agents; we titled the scheme Real-time Execution Checking and Deterrent Against Misbehaviour (RECDAM). Like the broader MASHIn vision, RECDAM is not biased towards protecting one or the other of the mobile agent paradigm major stakeholders. RECDAM was shown to be a very robust execution integrity scheme compared to existing execution integrity solutions for mobile agents. Two of RECDAM’s most admirable properties are that the execution integrity checks are performed immediately (i.e. real-time in-mission); and the execution integrity checks are performed completely independently of the agent user/home agent platform by neutral TTP entities - specifically MASHIn checking hosts and MASHs. The first property enables problems to be localised and more easily attributed to the source of the problem. The second property enables us to design a more flexible mobile agent execution integrity solution because it facilitates autonomous and secure agent mission completion performed
Chapter 9. A Framework for Transparent Confidence Scores in Uniting the Major Stakeholders

MASHIn checking hosts and MASHs are in an excellent position to provide reliable first-hand observations on fair and malicious stakeholder behaviour. In this chapter we discuss how this infrastructural design can be advantageously utilised to support the MASHIn vision. Two important benefits include: there is a strong, real deterrent against misbehaviour; and, previously mistrusting but well-behaved mobile agent paradigm major stakeholders can be transparently brought together for the initiation of useful work.

We start the chapter in Section 9.1 by taking a look at prior work on calculating mobile agent major stakeholder confidence scores (i.e. a measure of if they will behave honestly or maliciously). In Section 9.2, we specifically look at formulating a confidence score (on behalf of agent users) with respect to agent platforms in the MASHIn context. From the opposite perspective, we consider strategic utilisation of the MASHIn TTPs as an aid in protecting agent platforms from malicious incoming agents in Section 9.3. In Section 9.4, the flexible option to include second-hand opinions in the calculation of MASHIn stakeholder confidence scores is discussed - though, we note that due to the high quality of statistics collected first-hand by the neutral MASHIn TTPs, this should not generally be necessary. Finally, in Section 9.5 we round out the chapter with a summary of the intended knowledge transfer communicated herein, and the usefulness of the reliable confidence scores calculated transparently for both major stakeholders in supporting the MASHIn’s overall objectives.

This chapter includes and builds on material found in peer-reviewed accepted papers [i]-[iv] listed on page xxi.

9.1 Prior Reputation Systems Research

We located an early use of social control, by Rasmusson and Jansson, in decreasing attacks involving agents [156]. This work looked into discouraging dishonest or irrational agents in open markets by having the participants themselves responsible for the security, as opposed to leaving the security to some external or global authority. Social mechanisms do not deny the existence of malicious participants, but instead aim at avoiding interaction with them. Their work raised some pertinent recurring problem questions in user-issued reputation systems:

- How are new agents/entities introduced into the system, when they do not...
9.1. Prior Reputation Systems Research

have a reputation?

- Can an agent/entity build up a good reputation (either fairly or otherwise) and then abuse this to commit crimes (without punishment)? Conversely, what happens if an agent/entity is tarnished wrongly with a bad reputation and is hindered from doing business?

- Is any of this even legal?

Rasmusson and Jansson go on to discuss three possible mechanisms for social control. These are inspired by establishing opinions about other group members which are either local to each individual, distributed, or trusted to a third party. In the latter (i.e. TTP) case, they comment that reputation can be used in self-improving systems where the reputation corresponds to how well a service is performed. Later in this chapter, we will show how an application of this TTP concept is the basis of reliable confidence score possibilities in the MASHIn. We feel this TTP reporting of actors (agent platforms, for example, in the MASHIn context) whose services perform badly compared to others - thus encouraging use of services offered by actors with better reputation - is the process mechanism which is least vulnerable to improper use of reputations.

The effect of involving rumors with respect to the detection time of misbehaved nodes (in mobile ad-hoc networks), as well as the robustness of the reputation system against wrong accusations, is investigated in [24]. Mobile ad-hoc networks do not have the benefit of an infrastructure at their disposal to ensure correct behaviour, so any reputation model applied in this environment must be of the distributed variety. It is noted that relying exclusively on first-hand observations increases the time to detect malicious nodes when compared to an approach that uses reports from others (i.e. second-hand opinions). The latter reports, termed “rumours”, however can destabilise a reputation system when wrong observations (either accidental or deliberate) are provided.

As claimed by its authors, the first randomised live (Internet) experiment on the value of reputation was detailed in [160] - specifically on the eBay Internet auction site. After an auction ends on eBay, the buyer and seller each have the opportunity to rate each other’s performance by choosing either a 1 (positive), a 0 (neutral), or a -1 (negative) score rating. Users also have the opportunity to
leave a text comment and rated individuals can respond to recorded comments that they feel were unfair. The users’ net reputation scores, however, are calculated simply as the count of positive feedback scores minus the count of negative feedback scores. Strikingly, eBay feedback on the whole is very positive, with sellers only receiving 1% negative feedback and buyers only receiving 2% negative feedback. As the authors point out, it would be more helpful if the specifics of the negative ratings were more informative but no such mechanism is available. Also, a greater granularity for reputation scores (rather than just 1/0/-1) and/or sub-criteria for confidence score formations (e.g. timely service, paid in full, delivered goods were what was promised, etc.) would seem more practical in assisting in the decision of future commerce partners and one’s willingness to make higher or lower bids than otherwise would be the case. Over a number of experiments, the consensus reached suggests that a fairly effective reputation system may be good enough in this environment - thus, a reputation system that merely reduces the lure of cheating and poor behaviour without eliminating it may be sufficient to justify its incorporation.

There is a lot of good work being conducted on reputation in areas such as P2P systems and e-commerce contracts [25, 2, 13, 103]. Much of this work is out of the scope of the purpose of this chapter, which is to present a novel conceptual offering - that being, the application of confidence scores in mobile agent networks to assist both major stakeholders. The assistance is twofold: (1) Deterrence against misbehaviour, and (2) A transparent means to bring previously mistrusting major stakeholders into a position whereby a working relationship can be established. Whilst both of these objectives are admirable, the second one is very attractive as it eases a significant and ironic complexity with the practical advancement of the mobile agent paradigm - mutually mistrusting major stakeholders are fearful that their next step will be a losing one, but without the other they have no future [69].

A comprehensive security architecture for reputation systems is proposed by Ismail et. al. [92]; system entities are defined, and security services desirable for each entity are listed. The entities introduced are named as follows:

- The entity which is being evaluated is known as the Feedback Target (FT).
- The entity which evaluates the feedback target by way of feedback is known as the Feedback Provider (FP).
The entity which uses the confidence score produced to make a decision whether to proceed in a transaction is known as the Rellying Party (RP).

The Token Issuer (TI) manages the registration of FP and FT, records transactions made between FP and FT, and signs the token to produce a legitimate token.

The Collection Centre/Certificate Authority (CC/CA) collects the valid feedbacks and uses them to calculate confidence scores and then issue the certificates for the feedback target.

An adaptation of this security architecture, and its associated entity message interactions, will be introduced in the next section. We will see and explain why the formation of reputations in the MASHIn do not need as complex a reputation security architecture as outlined in [92], however we will highlight analogies where appropriate.

9.2 Formation of Confidence Scores on Agent Platforms

Two of the major aims in the design of the MASHIn (see Section 6.1.1), especially pertinent to this chapter’s contribution to the thesis, are:

- The problems in discovering appropriate business partners for mobile agent users and agent platform owners should be eased, and consequently a reduction in the key management problems for both sets of parties should be realised: This would assist the establishment and growth of a dynamic mobile agent community able to conduct business in a more secure fashion.

- Misbehaving entities should have their reputation lowered: The punishment is a strong deterrent against future malevolent actions.

As part of the MASHIn, as detailed in Chapter 8, we have devised an adaptation of Hohl’s reference states [86, 85] checking mechanism for increasing the confidence in a mobile agent’s execution integrity as it hops from one remote untrusted agent platform to another. The new, significantly more robust, scheme was titled Real-time Execution Checking and Deterrent Against Misbehaviour (RECDAM).
We will quickly review the basis of the RECDAM mechanism as it forms an important input to the base confidence scores of agent platforms that we foresee in the MASHIn.

9.2.1 Adapted Reference States Mechanism

Recalling RECDAM, with respect to the example depicted in Figure 9.1, an agent’s itinerary consists of three remote agent platforms (agent platforms 1, 2, and 3). The checking hosts - depicted in triangle-shaped figures - are MASH-delegated hosts inserted in the agent’s itinerary between remote agent platforms. Their main purpose is to verify the reference states claims from the previous agent platform, and if checked correctly forward the agent onto the next agent platform in its itinerary [71]. If a discrepancy is detected, they report this to the TTP MASH and also (by default) return the agent directly to the TTP MASH - the remainder of the agent’s itinerary is aborted.

The checking hosts should detect malicious agent platforms by not only checking for a discrepancy in the inputs from the previous agent platform leading to the claimed resulting state; they should also check a claimed initial state against the forwarded resulting state from the previous checking host. So, considering Figure 9.1 again, the TTP-MASH pre-posts its resulting agent state to checking host 1, checking host 1 pre-posts the checked resulting agent state from agent platform 1 to checking host 2, checking host 2 pre-posts the checked resulting agent state from agent platform 2 to the MASH. Thus, checking host 1, checking host 2, and the MASH respectively can detect discrepancies in the claimed initial state from agent platform 1, agent platform 2, and agent platform 3 respectively using the previous checking host’s forwarded resulting state.

The checking hosts have a primary role of checking the correct claimed resulting state from the previous agent platform’s execution of the agent for the TTP MASH. However, checking hosts also interpret ‘real’ agent callback classes (see Section 7.3.2) and monitor agent TTLs (see Section 6.5).

With the insertion of checking hosts into the agent’s itinerary after agent platforms, there could be a line of thinking querying the necessity of sending agents to agent platforms at all. Why not just send the agent to trusted hosts,

\footnote{To minimise itinerary completion times, the MASH can choose checking hosts with a close network proximity to the previous agent platform.}

\footnote{The agent user may specify an alternative course of action via the appropriate event-response methods in the agent’s secure callback classes (as discussed in Section 8.3.4.2).}
Figure 9.1: Adapted reference states agent movement for use in MASHIn RECDAM scheme.
like the MASH and checking hosts, and from there use the client-server model to communicate with non-trusted agent platforms?

We counter this argument with our long-term vision of a more trusting mobile agent community over time. One of the best methods to increase trust over time is to monitor the behaviour of past performances, as an entity’s past actions are a best indicator to the actions they are likely to hold to in their future behaviour. Thus, statistically capturing stakeholder behaviour in a highly accurate form is critical. From this data, confidence scores in stakeholders can be formed.

We show, in this chapter, how this macro-level thinking - considering both stakeholder interests, with their interests into the future at least as important as their current interaction - is supported by a solid framework for calculating stakeholder confidence scores in the MASHIn. Over an extended period of time and a credible reputation analysis to invest more trust in a target entity, a source entity may choose to incorporate a less security-rigid mechanism to guard their interests when interacting with that target entity.

With respect to the comprehensive security architecture for reputation systems presented in [92] (mentioned in Section 9.1), when considering agent platform base confidence scores in the MASHIn, we identify analogous entities. The agent platforms are the feedback targets. The inserted checking hosts (and the MASH at the start and end points in the agent’s itinerary) are the feedback providers. Agents (on behalf of agent users) are the relying parties. MASHs (or MASH-delegated entities) are the collection centres/certificate authorities, and similarly the analogous tasks of the token issuer falls ultimately to the MASH.

### 9.2.2 Events Involved in MASHIn Base Confidence Scores for Agent Platforms

In Figure 9.2, we complement the above abstract description of the formation of base confidence scores for agent platforms in the MASHIn with a message sequence diagram highlighting the communication between, and activities of, the key entities.

The agent user delegates mission protection of its agent to the MASH; the MASH is the key entity to each of the other entities. The MASH starts by determining a set of pertinent checking hosts for insertion into the agent’s pre-

---

3This is done transparently, as discussed in Chapter 10, for security reasons.
determined itinerary. It then executes the agent, performing any pre-mission initialisations.

Whilst the agent’s mission goal has not been deemed fulfilled (i.e. the outer loop, “loop (1)”), the agent visits a series of remote agent platforms performing part of its work. On the MASH, after a pre-determined itinerary, if the agent’s mission has not been met, the agent in collaboration with the MASH via agent callback code (privy only to the MASH) can determine a new itinerary - thus, maintaining the agent’s autonomy from its agent user. The strategic use of MASHIn agent callback methods was discussed in Chapter 7 and example utilities were mentioned in Section 8.3.4.

The MASH forwards the output state of the agent to the next checking host (this output state is the input state of the agent platform whose execution the checking host will verify). The MASH then sends the agent to the next agent platform to be executed there.

The agent platform, after executing the agent, is then required to send the agent to the checking host (along with the claimed input state of the agent and any inputs it provided for the agent). The checking host performs its primary responsibility, that of checking the agent’s state using the received information from the previous checking host/MASH and agent platform. (Subsequent agent platforms and checking hosts, in turn, replicate this procedure - hence the inner loop, “loop (2)”). The outer loop (“loop (1)”) is for dynamic itineraries, with the MASH supporting mobile agent autonomy via agent secure callback methods - whereas the inner loop (“loop (2)”) is for static, pre-determined itineraries.

Depending on the result of the execution integrity verification by the checking host (i.e. the alternative branch), the checking host forwards the MASH a message signifying either verification passed successfully with time-to-service-agent, or verification failed and why (with time-to-service-agent included in the message). The MASH then updates its statistics for the agent platform accordingly.

If there are more agent platforms to visit in the agent’s itinerary (i.e the optional branch), the checking host forwards the agent’s output state to the next checking host and then sends the agent to the next agent platform for execution there. The agent is executed at the agent platform, and the execution integrity verification process via the checking host is repeated. When the agent’s mission is concluded, the MASH securely returns the agent to its user.

The asterisk next to the first and last action in the sequence diagram highlights
Chapter 9. A Framework for Transparent Confidence Scores in Uniting the Major Stakeholders

Figure 9.2: MASHIn agent platform base reputation message sequence diagram.

Key (to Abbreviations):
- DAMP: Delegate Agent Mission Protection
- DCH: Determine Checking Hosts
- EA: Execute Agent
- POS: Forward Output State
- SAFE: Send Agent For Execution
- SAFIC: Send Agent For Integrity Checking
- PAIC: Perform Agent Integrity Checking
- RFRSC: Report Failed Reference States Checking
- USRFAP: Update Reputation Statistics For Agent Platform
- RTTPA: Report Time To Process Agent
- RCATU: Return Completed Agent To User
- MAPTV: More Agent Platforms To Visit (conditional)

* Agent User and MASH interaction should not be direct
the fact that the diagram is in fact a simplification of the real intended process (as discussed in the next chapter) whereby agent users and MASHs do not have any direct interaction. This is to assist in reducing the likelihood of DDoS attacks on MASHs (whose MASHIn internal IP address is not visible to outside entities), and to avoid any conflict of interest in the neutrality of MASHs with regards to the two major stakeholder entity groups.

### 9.2.3 Security Measures

The security measures which re-inforce the base confidence scores for agent platforms in the MASHIn are:

- MASHs form an explicit full-trust relationship between themselves and agents and agent platforms - thus, both stakeholders entrust the MASH (among a number of responsibilities) to not abuse confidence score calculations/issuings. Moreover, the legality of the MASH-issued confidence scores should never be an issue of concern.

- The employment of standard asymmetric cryptographic mechanisms for secure messaging between entities ensures channel confidentiality, data source authenticity, and data integrity.

- The MASH knows who the checking hosts will be for the agent, so the confidence score data feedback cannot be spoofed (because the MASH checks the digital signature of alleged checking hosts) - and, thus the confidence score data for an entity cannot be wrongly increased or decreased.

- MASHs and checking hosts (both trusted) have their system clocks synched. Therefore, statistics on agents’ time-to-service cannot be incorrectly claimed by agent platforms.

- The execution integrity check result, in our modified reference states checking mechanism, cannot be disputed by an agent platform due to the checking hosts who have no application-domain interest in the agent. And, if there is a reference check failure, the reason (pre-state failure, false input leading to false resulting state, etc.) is irrefutable.

Due to the strong detection and punishment nature of the MASHIn base reputation design, agents can confidently do business with (new) agent platforms
which do not have extensive statistical feedback from MASHs to formulate a robust confidence score.

The MASHIn framework for calculating stakeholder confidence scores is robust against the risk of an adversary’s ability to adopt or acquire multiple identities allowing the adversary to wreak havoc and abandon identities that are blacklisted while reappearing as someone new. Entities in the MASHIn, including target entities on which a confidence score analysis is formed, have to formally apply via a MASHIn-approved registration authority for a digital identity/public-key (as discussed in Section 6.2). As we discuss in Section 9.2.4, there is flexibility for relying parties to specify which personalised criteria are significant in reputation calculations - one could be the historical longevity of the target entity’s participation in the MASHIn. Couple these characteristics with the reliance on high-quality MASH-first-hand nature of observations in the MASHIn (discussed in Section 9.3), and it leaves the MASHIn largely immune from this general attack against a reputation system.

### 9.2.4 Formula for Calculating Agent Platform Confidence Scores

The agent user utilises transparently calculated confidence scores on agent platforms, with its own minimum confidence score requirements, to direct a facilitating MASH to transparently bridge a working relationship with itself (i.e. the agent user, and specifically its agents) and satisfactorily scoring agent platforms.

Even though the MASH (or more likely, in reality, a series of MASHs) accumulates agent platform statistics and trust relationship statements, flexibility should lie with the agent user to determine which statistics are used in the formation of reputation when making a personalised decision on interaction with an agent platform. For example, there could be an arbitrary number of possible properties used in the formation of confidence scores for agent platforms - we have touched on only two possible properties (percentage of RECDAM reference states failures and time-to-service-agents).

The agent user would choose which properties are significant in any personalised confidence score determination and to what extent they are pertinent - meaning what weighting should be attributed to each significant property. This feature is analogous to the rule-based approach - with application user defined tests and scores (i.e. weightings) - for filtering unsolicited e-mail messages in
SpamAssassin [198].

To give more concrete understanding, we will consider some possible statistics. A MASH may give a rating for a property between 0 and 100, with 100 being the best. Consider there are five potential properties expressed by a MASH with the following values $Property_1 \rightarrow 68$, $Property_2 \rightarrow 88$, $Property_3 \rightarrow 100$, $Property_4 \rightarrow 92$, $Property_5 \rightarrow 85$.

An agent (user) may be interested in only three of those properties ($Property_2$, $Property_4$, and $Property_5$). Furthermore, the agent would express weightings for the significant properties between 0 and 100, with 100 being complete significance. In our example, the agent provides its weightings for $Property_2$, $Property_4$, and $Property_5$ as 85, 100, and 100 respectively. A formula for a personalised rating utilising $n$ significant properties would be as follows (for significant $Property_i$ 1 to $n$):

$$\sum \left( \text{Significant}Property_i \times \text{Weighting}Property_i \right) \div n \times 100$$

Working with the provided values, we see:

$$\frac{(88 \times 85) + (92 \times 100) + (85 \times 100)}{3 \times 100}$$

$$= \frac{7480 + 9200 + 8500}{300}$$

$$\approx 83.93 \text{ confidence score}$$

The agent would, depending on its own minimum confidence score requirements for agent platforms, decide whether to conduct business with the agent platform (transparently via the facilitating MASH).

---

4SpamAssassin includes a large number of default tests and scores. However, the e-mail administrator has the flexibility to decide which tests are significant and what score they are weighted at in the calculation of filtered spam.
9.3 Anonymous Application of Agent User Properties for Protecting Agent Platforms

In the previous section, we discussed utilisation of reliably-sourced statistics on agent platform behaviour. The statistics are used as input by MASHs in the calculation of agent platform confidence scores. The confidence scores were seen to be personalised for agent users, as they nominate which available properties are significant and what weighting should be applied to each significant property in the calculation.

In this section we show how the MASHIn framework for calculating transparent confidence scores supports high quality reputation feedback for both major mobile agent system stakeholders - agent users and agent platform owners. Pivotal in this support are MASHs, as discussed in Section 9.3.1. We provide an example scenario of this reputation feedback for both major agent system stakeholders in Section 9.3.2. In Section 9.3.3 we discuss our success in meeting our two major aims for the strategic application of reputation in a mobile agent infrastructural context. Finally, in Section 9.3.4, we discuss a possible fee structuring model to support MASHIn major stakeholder confidence score formations.

9.3.1 Pivotal Role of MASHs

There is no reason why MASHs, acting in their TTP capacity, cannot and should not also gather highly reliable statistics on agents. These statistics can then be used, similarly, in personalised confidence score calculations of agent users on behalf of agent platforms. The use of these agent platform-personalised confidence score calculations via the MASH may be anonymous to the agent user (and, even, agent platform). Let us delve into this concept a little more.

Considering Figure 9.3, the agent user utilises the services of MASHs (and transparently, to be more specific, MASHs’ Trust Facilitation Module\(^5\)) to query which agent platforms it recommends are reliable - given some property benchmarks as discussed in Section 9.2.4 from the agent user - to fulfil a particular agent mission task.

On the flip side, agent platforms submit certain criteria (again, both a list of significant properties and weightings) to the MASH that agents should meet

\(^5\)Modules we foresee for the MASHIn were discussed in Section 6.1.3.
9.3. Anonymous Application of Agent User Properties for Protecting Agent Platforms

Figure 9.3: MASHIn entity trust interest interactions.

before MASHs forward agents onto their platforms to do work. MASHs are strategically positioned and capable of performing numerous first-hand observational tasks - and, thus, forming statistics - on agents. Plausible tasks include - but are not limited to - undertaking virus scanning, logic attack code filtering, excessive time-to-work per agent platform analysis, and filtering of non-authenticated (agent user) agents.

Reputation data is stored on the MASH entity component responsible for the Trust Facilitation Module. This database information will be a collection of many MASH (and checking host) sources.

9.3.2 MASHIn Reputation Example Scenario

An agent’s (let us identify it as agent/agent user AG_1) query to a MASH on reliable recommended platforms, given personalised significant properties and weightings, could return a subset list of recommended agent platforms. Let us say the agent platforms resulting from this query are \{AP_P, AP_T, AP_Y, AP_Z\}. However, the MASH will use knowledge of an agent platform’s target agent properties to filter a set of agents which should (and thus, also, should not) be granted
permission to do some work on their agent platforms. Let us say that the MASH has ascertained from these agent platform reputation criteria that agent platform AP_Y directs MASHs to not allow agents from AG_1 permission to be forwarded to its platform. Thus, the difference in these two sets (i.e. \{AP_P, AP_T, AP_Y, AP_Z\} - \{AP_Y\}) gives us the three MASH-recommended agent platforms (i.e. AP_P, AP_T, and AP_Z) for AG_1's query.

The fact that agent/agent user AG_1 has not been permitted migration access to AP_Y by AP_Y's owner, but calculated and managed by the MASH, remains undisclosed to the agent user (and can even remain undisclosed, if desired, to the agent platform) via the MASH. The intermediate MASH-calculated sets of potential agent platforms and agents need never be disclosed, thus anonymising entities for one or both of the major stakeholders.

Figure 9.4 pictorially highlights this example. Set 1 is the set of agent platforms the MASH recommends as trustworthy for AG_1 to do its work on, given the significant property inputs and weightings from AG_1 to the MASH in the calculation of confidence scores for agent platforms.

Set 2 is the set of agent platforms which direct the MASH to not forward agents from agent AG_1's user onto their agent platforms to do work on. To get Set 2, the MASH followed the following two steps (periodically, at some reasonable time-spacing):

1. Given significant property inputs and weightings from each requesting agent platform, devised recommended trustworthy agent users for each requesting agent platform in the calculation of personalised reputation ratings, and then

2. For each requesting agent platform, subtracted the personalised recommended trustworthy set of agent users ascertained in the above (1.) step from the complete universe of discourse of agent users.

Coincidentally, you can see that the MASH ascertained that agent platform AP_N directs the MASH not to forward agents from agent AG_1's user onto its platform to do work on, however because the MASH has not recommended AP_N to AG_1 as a trustworthy platform for AG_1's line of work, AP_N is irrelevant and did not even make it into the intersection of Sets 1 and 2.

Trust management for online environments is a relatively new research domain. Lack of trust is a real obstacle to the uptake of online services and mo-
9.3. Anonymous Application of Agent User Properties for Protecting Agent Platforms

9.3.3 Meeting Our Aims

One of the greatest challenges facing the proliferation of Internet mobile agents is the mutual mistrust between agent users and agent platform owners as both
have a lot to lose if attacked. We have shown that by strategically incorporating a TTP entity, the MASH, to leverage these concerns some significant progress can be made.

The trust explicit in MASHs, combined with the application of personalised confidence scores, enables the discovery of new business partners for both mobile agent users and mobile agent platform owners. The agent users and agent platform owners do not have to know any other entity (except indirectly MASHs via subscription to the MASHIn) to discover potential new business partners. The MASHIn confidence scores calculated have the following characteristics:

- *Personalised* because the relying party inputs which criteria in the formation of reputation are significant and to what extent they should be weighted;

- *Private* because neither the significant criteria and weightings, nor the ultimate decision on whether to engage in business with an entity, are available to any other party except the explicitly-trusted MASH;

- *Highly reliable* because MASHs uses first-hand observations (from neutral, unbiased TTPs i.e. other MASHs and checking hosts) as its foundation for calculating reputation on behalf of agent users and agent platforms;

- *Resistant to abuse* because no impact is felt if adversaries gang up unfairly in broadcasting either positive or slanderous confidence score statements on their peers. Stakeholder confidence scores will only align with their behaviour, and this behaviour will be accurately reflected in the reliable statistics collected by MASHs. The reliable analysis of major stakeholder behaviour strongly discourages stakeholders against the misuse of their business practice and penalises them accordingly in an unbiased, personalised reaction from its peers (in the form of lost business);

- *Transparent* because the confidence scores are calculated by MASHs (the two major stakeholders only specify relevant properties and their weightings), and new stakeholder relationships can be formed (without further interaction from the stakeholders) via these MASH personalised, private confidence score calculations.

The key management concerns for both agent users and agent platforms are significantly reduced due to the incorporation of MASHs. Agent users do not
9.4. Flexibility to Consider Second-Hand Opinions

have to know potentially thousands or millions of agent platform public keys to do business with securely. This key management handling responsibility is taken on transparently by MASHs and any other entities MASHs may subsequently utilise like public key certificate authorities. From an agent platform’s perspective the fact that more agents can find and do business securely with confidence on their platform increases their bottom line. Moreover, the authorisation of incoming agents is transparent to agent platforms based on personalised criteria expressed to MASHs who only forward agents passing acceptable confidence scores.

9.3.4 Fee-Charge Structure

MASHIn TTPs, especially MASHs (or more specifically the MASH-delegated entity administering the MASH Trust Facilitation Module), need to be remunerated for providing the stakeholder confidence score service.

MASHs can coordinate a billing charge per confidence score calculation based on the size complexity (e.g. number of entities to gather reputation statistics on) of the confidence score calculation being undertaken, the number of properties requested to benchmark in the calculation, the frequency/freshness of update statistics the requesting stakeholder wishes to base the confidence score calculation on, and the number of sources (MASHs, checking hosts, and second-hand sources). There could also be a charge based on the number of new stakeholders which are discovered by the MASH transparently for the requesting party to do business with.

The Transaction Non-repudiation Module (discussed in Section 6.1.3) is responsible for initiating stakeholder billing for MASHIn TTP security services delivered.

9.4 Flexibility to Consider Second-Hand Opinions

In any environment, for entity interaction to proliferate there needs to be a foundation for making positive trust decisions. Zolin et al. [242] encompass trust decisions with the following definition: “Trust is the deciding factor in a social process that results in a decision by an individual to accept or reject a risk based on the expectation that another party will perform to the individual’s expected
Our strongest basis for making such decisions is based on past-personal-direct experiences (i.e. reliable first-hand observations), but often we have not had any prior direct experience with an entity. Thus, we often rely on the observations of others (i.e. second-hand opinions) [24]. These second-hand opinions can form the basis of word-of-mouth trust decisions [148].

In this section, we first look at a “real-world” application of reputation ratings and its major shortcomings in Section 9.4.1. We discuss how the MASHIn reputation model is largely immunised from these shortcomings in Section 9.4.2 because MASHs use highly-reliable first-hand observations to drive confidence score issuings on behalf of the major stakeholders. Finally, in Section 9.4.3, we explain why and how the MASHIn reputation model can also support the inclusion of second-hand opinions in confidence scores if desired by relying parties.

### 9.4.1 Credit Rating Agencies in the Real-World

Let us consider the problems related to a ‘real-world’ application of non-first-hand observations as inputs into the process of calculating confidence scores. Credit record/rating agencies, like Dun & Bradstreet [56] and Baycorp Advantage [15], offer individuals or companies a credit risk analysis of potential business customers or partner entities (be they individual, business, or company entities). These credit record/rating agencies are consulted before we make new phone contracts with mobile phone providers, before we can take out a car or home mortgage, when we apply for a credit card or loan, and sometimes even in employers’ processing of job applications.

Credit providers (banks, building societies, government departments, etc.) supply information to credit rating agencies. It is not clear how much positive information credit providers forward to credit rating agencies on our sound use of credit - for example, Michelangelo had a credit account for $x and paid it off in the required term - but they certainly do report defaults in repayments.

Other companies then use these ratings to assess new applications for credit. Arguably the biggest problem with this credit rating service is that once a bad rating is established it is often hard to overcome - how do you get credit, so that you can satisfactorily pay it off to improve your record now that few, if any, parties are willing to offer you credit? The situation is compounded if the consumer credit history data fed from credit providers is erroneous. As this
information is second-hand to the credit record/rating agencies, they have no convenient means to ascertain the validity of such reports from credit providers\textsuperscript{6}.

Another problem arises when a consumer has no credit history and applies for a line of credit, especially a large line of credit like a home mortgage. In this case, from the lender’s perspective, the consumer’s lack of a responsible credit use history could be a significant barrier to granting the home loan.

We will now discuss if these problems afflict the MASHIn framework for calculating stakeholder confidence scores.

### 9.4.2 How the MASHIn’s Confidence Scores Fare

The above two confidence score problems pertaining to the credit record/rating agencies analogous application of reputation are not nearly as significant a headache in the context of MASHIn confidence scores. In the MASHIn model, MASHs collect data - either directly, or via trusted delegate entities (e.g. checking hosts) - which is fed into confidence score calculations of agents and agent platforms. The MASHIn calculated confidence scores for agents and agent platforms are more reliable due to direct utilisation of first-hand observations from reliable, unbiased MASHs and checking hosts (rather than word-of-mouth opinions). The business model of MASHs and checking hosts means they must remain unbiased and neutral to assure their income.

With regard to the second problem (i.e. the problem of a new entity with no reputable history), let us consider the interests of both of the major stakeholder interests (agent users and agent platform owners). With respect to the reputation of new agent platforms, agent users in the MASHIn context can sit comfortably because of the strong accountability procedures - including the incorporation of the neutral checking hosts utilised in the MASHIn RECDAM process (detailed in Section 8.3). Regardless of whether an attacking agent platform is new or long-standing, the subsequent checking host will detect attacks against the agent’s execution integrity (or service) and the responsible attacking agent platform will be held accountable.

Conversely, with respect to the reputation of new agents, agent platforms in the MASHIn context can still be confident of safe interaction. This is because the MASH will have an irrefutable record (from the current agent’s mission state)

\textsuperscript{6}Sometimes the reporting party may not even be a financial institution, for example a video store reporting a customer’s late-return or non-return of a hired item.
of any suspicions and/or discrepancies - like viruses, logic attack code, a non-authenticated source user - related to the agent.

9.4.3 The Option to Add-in Second-Hand Opinions

As we have highlighted, the very nature of MASHs’ first-hand observations on the behaviour of agent users (via their agents) and agent platforms means that the confidence scores formed by MASHs are highly reliable and unbiased. This work is both novel and can function alone in the formation of confidence scores for the respective stakeholders.

However, if agent users and/or agent platform owners want to also consider opinions from their peers (i.e. second-hand opinions) on the past performances of other agent users and/or agent platforms, this can be added into the equation for formulating personalised confidence scores.

Significant work on agent-to-agent reputation has been documented in the existing body of literature [24, 148, 193, 102]. One of the better pieces of work in this regard is the forming of dynamic new relationships utilising word-of-mouth opinions via a facilitator as discussed in [148]. With respect to this existing work, the generic use of the term “agent” could be construed either as an agent user or an agent platform.

Whether second-hand opinions would be uploaded and stored on MASHs would be up to the agent user or agent platform entities providing the advice (i.e. feedback providers). If these statements are stored on MASHs, new relationships could be transitively formed as required (either with explicit permission granted by an advising entity to a seeking entity, or via anonymous private facilitation by the MASH).

Incorporating second-hand opinions into the general formula (weighted-average of significant properties) presented in this chapter would be a trivial inclusion as follows: which second-hand opinions are significant and their weighting would be expressed to the MASH via the relying party. The MASH proceeds then to calculate the personalised confidence score using the weighted-average of significant properties algorithm supplied in Section 9.2.4. Second-hand opinions could be accumulated by MASHs, along with MASH first-hand observations, where the designer deems a mobile agent network with checking hosts as an unnecessary complexity.

Thus, first-hand observations or second-hand opinions could be used solely,
or a union of both first-hand observations and second-hand opinions can be used in the formation of MASHIn confidence scores.

## 9.5 Summary

There are no definitive answers or easy solutions for the protection of mobile agents deployed onto the Internet. The nature of Internet mobile agents, their flexibility to roam and work on untrusted agent platforms, poses great dangers for their misuse - especially from malicious hosts. Mobile agent platforms, indeed, also face great losses if they are attacked and compromised by mobile agents and other entities. This mistrust complexity, ironically involving the two major stakeholders in the mobile agent paradigm, remains a significant barrier to the advancement of mobile agents geared for work on an electronic marketplace of potentially thousands or millions of agents platforms.

Whilst the bulk of other researchers working in this field have focused on protection for only one stakeholder, and usually on measures to achieve one property (like confidentiality or integrity of an agent in execution, or conversely protecting the agent platform from access violations or hijacking resources), we have taken a higher “macro-level” approach to addressing security concerns plaguing the prospects of Internet mobile agents. We have done this because we feel it is the best course of research for the long-term prospects of the mobile agent paradigm - without a higher-level approach to bringing mutually mistrusting stakeholders to the bargaining party, the “micro-level” mechanisms will go largely wasted.

In the two previous chapters we documented highly-effective code/data privacy and execution integrity solutions, with abbreviated titles SCC4MAP and RECDAM respectively. Both schemes fitted in well within the MASHIn framework and afforded protection for both major stakeholders - mobile agent users and mobile agent platform owners.

In this chapter we introduced a novel macro-level method of transparently calculating confidence scores in a mobile agent infrastructural context. In other research domains, it has been shown that the utilisation of reputation can be a strong deterrent against misbehaviour, but needs careful management so as not to disadvantage the introduction of new entities who do not have a reputation into the infrastructure or to risk the misuse of reputation against adversaries (by, for example, ganging up on an entity unfairly by spreading false, slanderous
Chapter 9. A Framework for Transparent Confidence Scores in Uniting the Major Stakeholders

Our TTP-based application of reputation in the MASHIn uses highly-reliable first-hand observations from the explicitly-trusted MASHs and checking hosts. The use of TTP first-hand observations in this fashion (over second-hand opinions) are not only more reliable in assisting trust decisions, but it also means that the impact of rumours is mitigated. Moreover, if an entity does misbehave, the MASH is strategically positioned to punish the offending party (via a lowering in their confidence score statistics, and possibly some other corrective penalty). Stakeholders are not discouraged from doing business with new entities not having extensive statistics for formulating a confidence score on, as MASHs (also utilising checking hosts) have strong detection mechanisms for keeping attacking entities accountable.

The requesting party, an agent user or agent platform owner, has complete flexibility into the criteria used to calculate personalised reputation ratings by a MASH. Not only which properties (i.e. behavioural characteristics on an entity) should be significant, but the extent to which they are of significance/weight are passed onto the MASH in the formation of personalised confidence scores. We have shown how, in submitting this criteria to MASHs, agent users and agent platform owners can transparently discover new business partners via the MASH. The complexity of cryptographic key management and authentication of new business partners is transparently handled by MASHs.

At least three issues need to be explored with the conceptual framework for calculating confidence scores presented in this chapter, for those wishing to implement a prototype or production MASHIn system. Feedback data from checking hosts (and other sensors\(^7\)) in the MASHIn network could be transport or computationally expensive due to large amounts of data shifting and acting as inputs into (potentially frequently refreshed real-time) calculations. How to manage this efficiently could prove challenging. Determining who can be an ultimate source (i.e. head entity potentially in a tree-structure) of trust in the MASHIn, and defining cross-community relationships - a problem with generic global PKI models (see Section 6.2 for related discussion) is an issue potentially limiting the MASHIn’s heavily TTP-reliant model. Finally, compromise of one or more prominent MASHs could jeopardise the credibility of MASHIn confidence scores - thus, very strong precautionary methods must be designed and adhered to.

\(^7\)For example, strategically-deployed network virus scanning devices.
Future work can look at more mechanisms for collecting first-hand observations from MASHs and checking hosts including a variety of agent execution integrity checking mechanisms. The content and frequency of data communicated between MASHs and checking hosts can also be studied. Though started in the next chapter (i.e. Chapter 10), methods for protecting MASHs (and checking hosts) should also be explored more by others in future work, as a lot of proposed security functionality in the MASHIn rests with these two crucial TTP entities.
Chapter 10

Measures to Consider in Protecting MASHs and Checking Hosts

Science is organized knowledge.
Wisdom is organized life.

Immanuel Kant

In the first half of this thesis (i.e. chapters 2-5 inclusive) we painted a picture detailing how the threat countermeasures and perspective taken by investigators researching mobile agent security has largely not touched on a major security issue with advancing this field. The fact that existing mobile agent security countermeasure approaches largely do not take into consideration the security interests of both major stakeholders (i.e. mobile agent users and mobile agent platform owners) is a large void in this research field, as one of the main problems in advancing the mobile agent paradigm is the ironic mistrusting major stakeholder problem. Both (potential major stakeholders) are fearful that their next step could be a losing one, so they are hesitant to engage in interaction with the other in the first place - but, both stakeholders need each other for the mobile agent paradigm to be viable on a wide Internet scale. We, thus, concluded that a more holistic approach addressing both stakeholder security concerns was needed within a framework enabling interoperable and extensible security services.
In the second half of this thesis (i.e. chapters 6-9 inclusive) we presented a more encompassing approach to advancing the prospects of the Internet mobile agent paradigm, by focusing on the security interests of both paradigm major stakeholders. We devised a novel infrastructural-level conceptual solution titled the Mobile Agent Secure Hub Infrastructure (MASHIn). The MASHIn provides a number of abstractions and techniques including support for flexible access control of resources, mobile agent data and code privacy, mobile agent execution integrity, and a transparent method to calculate stakeholder confidence scores and bring unknown - and, by default, mistrusting - stakeholders to working relationships. All presented MASHIn security strategies considered the security interests of both stakeholders, with no bias in design shown towards either stakeholder. Pivotal in these endeavours was the strategic utilisation of well-known entities (including RAs, CAs, and AAs) and new TTP entities (specifically MASHs and checking hosts).

In this chapter we touch on the importance of protecting these new TTP entities, as their correct functioning is vital to the security interests of both major stakeholders. We provide an overview of some important measures which may be used to protect these TTP entities, and mention the state-of-the-art techniques in the various related research fields. We begin in Section 10.1 by discussing distributed deployment and backup procedures. In Section 10.2 we discuss mandatory access control and research directions in securing operating systems. In Section 10.3 we discuss the concept of abstracting these new TTPs in a Virtual Private Network (VPN) so that they are accessible only to readily identifiable entities within the MASHIn. Utilising defence-in-depth firewall design is discussed in Section 10.4, and measures to counter the threat of distributed denial-of-service attacks are discussed in Section 10.5. Trusted computing initiatives are then discussed in Section 10.6, before rounding the chapter off with a summary in Section 10.7.

This chapter includes and builds on material found in peer-reviewed accepted paper [ii] listed on page xxi.

10.1 Distributed Deployment and Backup

As pointed out in Section 6.1.4, it is not intended that the MASHIn model depend on only one or a few dedicated servers to deliver MASHIn security services for
both major paradigm stakeholders. We recommend that there be distributed deployments of MASHIn TTPs and TTP components so that the MASHIn:

- Is not crippled by a single point-of-failure.
- Utilises localised TTPs, especially checking hosts and MASHs, to the agent platforms that an agent does its work on (to minimise transport costs where possible).
- Supports a large-scale network and large number of participating major stakeholders (via their agents and agent platforms).
- Supports automated off-site backup procedures, transparently processed at appropriate periodic intervals.
- Supports fault-tolerance by replicating important information - for example mobile agent ‘real’ callback class privacy, execution integrity, and TTL processing information across checking hosts.
- Serves certain jurisdiction and/or locale boundaries, as needed.

Furthermore, stakeholders - in particular mobile agent users - may rely on certain TTP hardware configurations (for assisting in assuring the security and safety of mobile agent missions). This can be transparently supported in the MASHIn, and still avoid a TTP conflict-of-interest involving either major stakeholder, as discussed in Section 10.3.

We also briefly mentioned likely coupling of major MASH software modules that we foresee for the MASHIn in Section 6.1.4. An important point to rehighlight in this regard is the fact that MASHs are not expected to directly handle all MASH-abstracted security responsibilities themselves. Indeed, this would bestow MASHs with multiple complex functions and an over-burdening processing load. MASHs will, more simply, offload security responsibilities to more specialised providers, maintaining a uniform abstraction representing the glue between these delegated entities, themselves, and the two major paradigm stakeholders.

Aligning with the deemed criticality of the data stored and/or served on MASHIn TTPs, a range of the state-of-the-art backup, recovery, and archiving technologies would be adopted. Fully-redundant and clustered data centres
having high speed and fail-over reliable Internet connections, coupled with multi-tiered network security measures are available [41, 117, 138]. Nothing less than a 99.9% guaranteed uptime for these managed data storage and monitoring services can be mandated.

Real-time transaction-level replication database support, including synchronous transaction replication functionality for fail-over clustering in Oracle’s Enterprise DBMS [151] and Microsoft’s SQL Server [131] DBMS, is common today and avoids many of the data conflict inconsistencies that came with replication procedures of past years. Synchronous replication solutions for open source databases are well-advanced also [9, 67].

Synchronous replication support is an important property in ensuring that no mission-critical transactions and data are lost in the event of a server going down for whatever reason. When coupled with high-quality monitoring and recovery mechanisms, like those pointed to above, the continuity of an essential security service can survive many catastrophic events affecting a server, site, or region.

10.2 Mandatory Access Control and Secure Operating Systems

Mainstream operating systems are increasingly being exploited by malicious code exploits including viruses, worms, trojans, and buffer overruns. With increased interconnectivity, standard computers utilising mainstream operating systems have never seen more security incidents associated with them [27]. Core major problems with their design are their dependence on discretionary user access control decisions and their vulnerability to completely being taken over by an adversary due to privilege escalation and/or breach to gain superuser/root/administrator user level access. It goes without saying that equipping MASHs and checking hosts with mainstream discretionary access control operating systems is not a valid option.

The question to ask then is if “hardened” and/or “trusted” operating systems offer sufficiently solid foundations for MASH and checking host operating system protection. A hardened operating system is one that has been locked down to prevent attacks; how well the system is locked down is largely subjective and the methods vary from company to company [22]. An example of a hardened operating systems is OpenBSD [149], with much of the common default open
10.2. Mandatory Access Control and Secure Operating Systems

network ports and services associated with other Linux/Unix distributions being removed. A trusted operating system is developed from a formalism that was first theorised and proven over a decade starting from the mid-1980s in research led by the National Security Agency (NSA) and the Department of Defense (DoD) in the United States [4]. An example of a trusted operating system is Sun Microsystem's Trusted Solaris [188].

Access controls in trusted systems are significantly more fine-grained than those in mainstream operating systems, limiting the damage that an attacker can do given they take control of a running process. Moreover, trusted systems often have been certified against some assurance certification criteria, such as the Common Criteria [141], which embody “best practices” in the security of IT products. However, there still remains some conjecture among security experts about how best to define Common Criteria specifications for the security of embedded operating systems and really how much added security one gains from an operating system meeting directed protection profiles at some evaluation assurance level [238, 177]. Even achieving EAL7 - the highest possible evaluation assurance level (which no publicly-available operating system has attained) - only requires a “semi-formal” specification of the low-level implementation. Formal specification of complex software is, itself, a long way from being a solved problem.

Another category of trusted operating systems adds mandatory access control (MAC) primitives to their design. Mandatory access control enforces corporate security policies that deal with the sharing of data by comparing the sensitivity of the resource (e.g. file or storage device) with the clearance of the entity (e.g. user or application) [93]. As opposed to discretionary access controls (DAC) that rely on users to specify a security policy, mandatory access controls rely on centrally administered policies that are imposed on every data item in the system throughout its lifetime [79]. DAC access decisions are based only on user identity and ownership, ignoring other security-relevant information such as the role of the user, the function and trustworthiness of the program, and the sensitivity and integrity of the data. Typically, only two major categories of users are supported by DAC mechanisms, completely trusted administrators and completely untrusted ordinary users. Many system services and privileged programs must run with coarse-grained privileges that far exceed their requirements, so that a flaw in any one of these programs can be exploited to obtain complete system access [120, 22].
Adding MAC mechanisms is important to the core robustness of the security architecture of operating systems, as it can address many of the vulnerabilities associated with DAC - including the abovementioned superuser escalation problem, and malicious software being able to change user and group objects as it chooses because of the impossibility of enforcing a system-wide security policy [120]. When properly configured, MAC operating systems can confine user programs and system services to a minimum amount of privilege to do their jobs so that even if these programs and services are compromised, the system’s security as a whole is not jeopardised. This fine-grained localisation of security problems would be critical in ensuring MASHs and checking hosts can deliver their security services on behalf of both major stakeholders.

A trusted path, a mechanism by which a user may directly interact with trusted software - initiated by either party but not initiated by other software\(^1\), is an important safeguard in ensuring malicious software does not impersonate trusted software to the user or impersonate the user to trusted software. Without operating system support for mandatory security and trusted path, higher-level application-space mechanisms for access control and cryptography cannot be implemented securely - the case in most mainstream operating systems today [122].

TrustedBSD [220] and SELinux [139] are open-source secure operating system initiatives with different objectives. TrustedBSD provides a set of extensions to the FreeBSD operating system - with an aim towards incorporating many of its contributions back into the FreeBSD base distribution. The extensions target the Common Criteria with more fine-grained discretionary access control and capabilities, extended file attributes and auditing, and mandatory security with a variety of access control models supporting strengthened integrity and confidentiality requirements. By targeting the Common Criteria, TrustedBSD - as its name suggests - aims to become an accredited “trusted” operating system, but it is unclear how successful it has been in this regard. From its website, no new developments on TrustedBSD have been published since mid-2003.

SELinux (“Security-Enhanced Linux”) is a more mature and ongoing project concentrating purely on the design of a solid and flexible mandatory access control model for the Linux kernel. The prototype kernel contains architectural components originally developed to improve the security of the Flask operating system [184]. SELinux provides general support for enforcing many kinds of

\(^1\)A protected path is a more generalised concept - ensuring mutual authentication and protecting against spoofing in a networked system.
mandatory access control policies, including those based on the concepts of type enforcement, role-based access control, and multi-level security. SELinux was not started with the aim of becoming a “trusted” operating system nor does it claim to be one today. Its primary focus was to incorporate a solid mandatory access control framework into the standard Linux kernel and to demonstrate secure applications of this research. SELinux may move towards some “trusted” accreditation if efforts in other areas (e.g. auditing and documentation) are combined with its primary objective [139].

The SELinux kernel enforces mandatory access control policies that can confine user programs and system servers to the minimum amount of privilege they require to do their jobs. This reduces or eliminates the ability of these programs and daemons to cause harm when compromised via buffer overflows or poor configuration. The confinement mechanism operates independently of traditional Linux access control mechanisms. SELinux has no concept of a “root” superuser, and does not share the well-known shortcomings of the traditional Linux security mechanisms, such as a dependence on setuid/setgid binaries. SELinux supports flexible separation policies for restricting access to classified data, containment policies for minimising damage caused by malicious code, integrity policies for protecting against unauthorised modifications of data and applications, and stronger controls over process initialisation and inheritance [139, 120, 121].

Another important research area concerning mandatory access control which has largely not received much attention is that of mobile-code platforms such as the Java Virtual Machine (JVM) and the .NET Common Language Runtime (CLR) which have implemented far weaker security models that do not reliably track ownership and access permissions for individual data items. Virtual machines such as the aforementioned two perform a number of static and dynamic checks to ensure a basic level of code security, which is useful in preventing most viruses and buffer overflow attacks. However, higher-level policies that depend on program state cannot be specified, for example “disallow transmitting on network after reading from the local filesystem” [79]. Moreover, real-time automated tracking and restricting object resource consumption in virtual machines, such as the JVM, is a huge problem [168, 133, 82, 182].

Implementing mandatory access controls in virtual machines will strengthen the security model and address some of these security vulnerabilities which presently threaten the robustness of application-space safety in virtual machines.
A promising start by Haldar and Franz [79] adds mandatory access control to an existing JVM at the granularity of objects. The enforcement mechanism is separated from the specification of policies, allowing flexible specification and enforcement of a wide range of policies. In related work, they also use mandatory access controls for “semantic” remote attestation of virtual machines [78]; this work could enable a remote party to have some means of constraining how its data is handled by a trusted virtual machine. Remote attestation is discussed more in Section 10.6.

### 10.3 Virtual Private Networking

Up until now, to keep matters simpler when explaining things we have described a MASHIn agent user as directly interacting with a MASH - with the implication that the agent user can choose a specific MASH to delegate agent mission protection to. Whilst it is true that an agent user should and can request the services of a MASH offering certain security services, the choice of an exact MASH to employ should be avoided. In Section 6.1.2, we gave notice to the reader of this presentation simplification. An example of this simplification can be seen in Section 6.4, specifically in step 7 of the MASHIn agent (user) authentication and anonymisation processes described therein, associated with Figure 6.5. The following is stated there: “The agent user forwards on their (signed) agent and associated agent attribute certificates to a MASHIn Mobile Agent Secure Hub (MASH) for protected mission execution”. This direct agent user and MASH interaction should not be the actual process to employ in the MASHIn for at least two important reasons:

- MASHs are the most important abstraction in the MASHIn, vital to both major stakeholder security interests. To dissatisfied entities not interested in the MASHIn, MASHs would be an obvious target for distributed denial-of-service attacks and active (e.g. injecting malicious code on a MASH) compromise attacks. Making MASHs “invisible” (i.e. directly unreachable) on the Internet would significantly mitigate this risk.

- A prominent ingredient in building reliable, non-biased major stakeholder confidence scores in the MASHIn is the neutrality of delegated MASHs (and
subsequently the sub-delegated checking hosts)\(^2\). Removing the possibility of an agent user directly choosing any one specific MASH for agent mission protection reduces the potential implied or explicit bias towards the agent user from a MASH’s perspective.

In Section 6.1.4 we highlighted the importance of not only having a number of MASHs and checking hosts, but deploying them on a wide geographical basis. Important reasons given there include dependence on a single point-of-failure, decentralised TTP location and management to reduce transport costs, that the MASHIn should be capable of supporting possibly millions of agents (and thousands of agent platforms) at any one time, and that MASHs and checking hosts could conceivably serve particular jurisdiction and/or locale boundaries.

We now present a more realistic, dependable architectural model for the MASHIn proposal. We do not wish to claim or imply that it is a complete readily-deployable solution; presenting it does, however, help to better conceptualise precautionary measures for important entities in the MASHIn and lays a better foundation for future work on moving the MASHIn concept towards a real-world working solution. The ensuring discussion refers to Figure 10.1.

Agent users will make a request for utilising the MASHIn scheme via a Virtual Private Network (VPN) connection. As a possibility, the robust Internet Key Exchange (IKE) protocol [81, 143] could be used as an authenticator and negotiator of an IPSec VPN connection here. This protocol would verify the agent user as someone who should be allowed to start encrypted communication with the device in question, in our case an external MASHIn network firewall. A solid way of authenticating users in IKE is by employing digital certificates and public-key cryptography. MASHIn entity registration processes were discussed in Section 6.2, specifically the off-line process whereby MASHIn entities (including agent users, agent platforms, MASHs, and checking hosts) must apply to a MASHIn-accredited Registration Authority (RA) for a MASHIn public-private key-pair which is thereafter certified by a MASHIn-accredited Certification Authority (CA) who coordinates the certificate’s life-cycle. Only requests from authenticated agent users will be able to pass the external firewall into the de-militarised zone (DMZ).

\(^2\)A framework for calculating major stakeholder scores in the MASHIn was presented in Chapter 9.
Figure 10.1: Basic MASHIn network architecture proposed.
In a DMZ area, there will be a bank of anonymisation servers behind a load balancing component. It is the responsibility of the anonymisation server to check that the agent user has been granted the appropriate authorised roles for its mission, and to anonymise the agent via a delegate proxy certificate. The utilisation of Attribute Authorities (AAs) in a privilege management infrastructure supporting RBAC to MASHIn agent platform resources was discussed in Section 6.3. By the time an agent is delivered to a MASH for the first time it has been anonymised (i.e. the MASH does not know the agent’s user) by a delegate proxy certificate granted from the anonymisation servers. The basic anonymisation process was explained in Section 6.4 - once again, with the simplification that the agent user and MASH were in direct interaction. In that explanation, the MASH performed the anonymisation of the agent (user) in respect to the agent’s work on agent platforms. However, in the MASHIn architecture depicted in Figure 10.1, the anonymisation servers in the DMZ are performing the anonymisation of an agent user’s identity with respect to agent platforms and the MASH assigned mission protection of the agent.

In practice, there will likely be other servers in the DMZ, for example servers to report on the location and status of deployed mobile agents. Servers to accept new agent user and agent platform owner subscriptions from MASHIn-accredited registration authorities will be required, as will servers to manage entity public-key updates from MASHIn-accredited certification authorities. The proposed MASHIn architecture in Figure 10.1 is to be used only as a base guide for conceptualising the proposed MASHIn security framework for mobile agents, and is expected to undergo a number of refinements into the future as the MASHIn evolves to a real-world deployable model.

The internal firewall permits valid data generated from the servers in the DMZ to pass through to a non-Internet facing (i.e. inaccessible from the Internet) MASHIn wide-area network (WAN) Intranet. This MASHIn WAN Intranet comprises at a minimum a number of MASHs and checking hosts, possibly organised into a set of LANs dispersed by geographical/jurisdiction boundaries. The servers in this WAN Intranet have private IP addresses; it is expected that network address translation and proxying to the outside world is provided via the internal firewall and appropriate machines in the DMZ respectively. As more MASHs and checking hosts are required, dependent on subscriber numbers, to service the MASHIn objectives more MASHs and checking hosts can be added to
this heavily-firewalled WAN Intranet environment.

The assigned MASH would coordinate mission-protection of the agent, starting by choosing a number of appropriate checking hosts for insertion into the agent’s itinerary. The agent travels to a number of agent platforms (each of which is responsible for protecting its own network) located on the Internet in between travel to MASHIn MASHs and checking hosts, as explained in Section 6.5. It is expected that these travels will be protected by virtual private network connections, as well.

Similar to the agent user’s perspective, it is important to note that from an agent platform’s perspective it does not know which MASH and checking hosts are being employed for an agent’s mission, reducing the likelihood of bias arising from either of these MASHIn TTPs and giving more credence to the reliability of MASHIn-calculated major stakeholder confidence scores.

We wish to point out, again, that Figure 10.1 is only a basic indication of the foundations from which a MASHIn network architecture can be built and improved on. To reduce the risk of overloading the external firewall/DMZ/internal firewall path shown in that architecture, solutions for incorporating multiple connections into (and out of) the MASHIn non-Internet facing WAN Intranet must be investigated. Having only one internal/external path would result in that path being a choke point in the network architecture - from both legitimate and malicious network traffic - and seriously jeopardise the continuity of mission-critical security services offered to both major paradigm stakeholders by the TTP servers inside the MASHIn WAN Intranet. Matters concerning WAN hardware, routing methods, and protocols for managing WANs are out of the scope of the purpose of this thesis. A good reference on planning mission-critical networks, including design decisions for WANs is the book written by Liotine [119].

10.4 Defence-in-depth Firewall Design

Castles are a non-contemporary example of a defence-in-depth strategy. For example, to gain access to the princess, one would have to cross the moat and bash down the big door; then you are in a sallyport area (similar to an airlock), you have external control doors, an assembly area and an interior set of doors and only one set of doors should open at a time. So even if you break down one set of doors, you have to cram into the assembly area, there may be ports where
the insiders can shoot at you and you have another set of doors to go [143].

The non-Internet facing MASHIn network needs to be protected by a robust, but flexible strategy that will ensure mission-critical services offered by the strategic MASHIn TTPs - namely MASHs and checking hosts - go unimpeded. The MASHIn firewall perimeter will need to employ state-of-the-art technologies and (just as importantly) best-practice management procedures to minimise the threat that an attacker can compromise the security of the DMZ. Running the minimal number of services (as required) in the DMZ, and breaking up the DMZ into several “security zones” associated with particular services housed in the DMZ is one such strategy. Keeping aware of the shrewdest intrusion detection and prevention products and procedures will also be important.

We expect employing outsourced professional expertise for real-time managed security monitoring, such as that offered by Counterpane Internet Security [44], should be seriously considered for the MASHIn. From a research perspective, formal methods for analysing security engineering processes such as firewall and network intrusion detection system configuration updates [209] is a very challenging but important area of investigation.

Learning lessons from faults in traditional defence-in-depth strategies will be critical, for example problems related to backdoors and secret passages proved fatal for the defences of more than a few castles, as did the fact that they were stationary targets [143]. With regard to the MASHIn, a series of carefully thought out network traffic filters and/or procedures would need to be designed and maintained so as to minimise the chances that legitimate backdoors\(^3\) are robustly secure (i.e. from illegitimate outsiders). Furthermore, state-of-the-art management of internal threats/attacks is required - such as generally not allowing traffic to source from the internal MASHIn WAN, and applying mandatory access policies so that compromises cannot be discretionarily propagated.

We have already mentioned the importance of redundancy in terms of both backup/replication/failover procedures and network paths for legitimately accessing MASHIn security services. Redundancy is also an important property in countering distributed denial of service attacks, a threat we discuss in the next section.

\(^3\)Legitimate backdoors are sometimes needed by system administrators for alternate access in the event of network emergencies.
10.5 DDoS Countermeasures

A serious threat to any interconnected system or network is that of a Distributed Denial of Service (DDoS) attack, a co-ordinated attack on the availability of services launched indirectly through many compromised computing systems [183]. Indeed, it can be argued that the Internet as a medium for delivering reliable network services is under direct threat if architectural revisions are not studied [80, 54, 127].

The real problem with countering DDoS attacks is the fact that traditional distributed systems defence mechanisms such as demanding strong authentication and employing firewalls are less effective in countering DDoS attacks. This is due to the sheer volume of network traffic generated in co-ordination by injected agents into compromised third-party systems, coupled with the de-centralised and autonomous/anarchic management of systems connected to the Internet. The target of the DDoS attack is referred to as the “primary victim” and the machines that the attacker compromises to inject the distributed attacking agent software on are referred to as the “secondary victims” or “zombies” (because their participation in an impeding co-ordinated attack is unbeknown to them) [183].

Though DDoS attacks have been recognised since the TRIN00 attack tool was used in 1997 [8], it appears that researchers have only very recently attempted to classify the types of DDoS attacks and available countermeasures. Specht and Lee [183] have produced a taxonomy of DDoS attacks, tools, and countermeasures. A more comprehensive taxonomy and discussion on DDoS attacks and countermeasures is presented by Mirkovic and Reiher [135].

The countermeasures currently available for mitigating DDoS threats are partial solutions at best, as there is no comprehensive method to protect against all forms of DDoS attacks [183]. Existing DDoS countermeasures are sub-grouped by Mirkovic and Reiher [135] under the categories of preventive or reactive activity level, degree of cooperation (autonomous, cooperative, or interdependent), deployment location (victim network, intermediate network, or source network), and attack response strategy (agent identification, rate-limiting, filtering, or reconfiguration).

Installing software patches and applying built-in defences are in the category of DDoS countermeasures for individual users to detect/prevent secondary victims, identified by Specht and Lee [183]. Potential secondary victims must continually monitor their own security measures, and make sure that they are not indirectly
sending agent traffic onto the Internet. These measures for the average novice user (a technically-illiterate home user with a broadband Internet connection such as a cable or DSL modem) are daunting. It has been suggested that beneficiaries of upstream DDoS preventative measures such as ingress filters could remunerate the domain who is acting responsibly - to encourage this safe practice, despite the fact it consumes valuable router resources and reduces the overall routing performance [116]. Perhaps this idea could be slightly re-applied to pose economic penalties on service providers who do not act responsibly in terms of producing authenticated, legitimate Internet packets. Taking this thought a little further, perhaps it could be possible to impose punishments on operating system providers whose products do not provide default, reasonably safe mechanisms to prevent insecure code being injected into their systems.

The merits of educating future IT security engineers by introducing them to a controlled DDoS disaster, inspiring them to creatively search for a solution that will prevent or appropriately react to such an attack is presented by Bolz, Romney, and Rogers [21]. Knowledgeable and flexible application of mitigation strategies is imperative to handle ad-hoc, new DDoS attacks in the wild [21, 135].

It is argued by Lee and de Veciana [116] that solutions to thwart DDoS attacks should also be distributed to be effective, and block attack packets as close as possible to the attack nodes; they present a multicast-based filtering and tracking service framework to defeat DDoS attacks. If attacks are assumed to be infrequent, their model can achieve more efficient use of network resources than proactive solutions. When no attacks are ongoing, only a multicast session needs to be maintained, without overheads associated with a filtering operation.

Yan, Early, and Anderson [231], based on a sound cost/benefit analysis of the motivating factors in initiating and defending DDoS attacks, propose an interesting dynamic replicated network bandwidth solution that aims to defeat DDoS attacks. The business model is that ISPs run a number of Xenoservers on which they rent capacity to service providers. A Xenoserver [158] is a service running on top of an operating system built to ensure quality of service guarantees, and thus may be well-suited to addressing incoming DDoS attacks. The quality of the service is monitored and, if it starts to deteriorate as a result of a surge in demand, the web site is at once replicated to other Xenoservers whether locally hosted or in other ISPs. In the latter case, distributed DNS techniques can be used to eliminate bottlenecks. As ISPs get paid for the resources they supply, there is
an incentive for them to install and maintain a sufficient number of Xenoservers to meet whatever service levels are set in their contracts with the customer. It is claimed that since the ISP hosting the attacked system can rent additional capacity as required, it needs less installed capacity to cope with everyday demand fluctuations and should in any case be able to insure against the costs of absorbing particularly severe DDoS assaults. Xenoservers could also be hosted in closed private network environments such as those held by governments and large corporations [231].

A set of changes to the Internet architecture are suggested by Handley and Greenhalgh [80]. They propose dividing the IP address space up into client and sever specific addresses, in association with enforcing rules restricting how those addresses can be used (e.g. preventing servers from making outgoing connections to reduce propagation of Internet worms). This would result in more manageable interactions such that the vast majority of desirable network interactions are allowed, and the large number of undesirable network interactions are disallowed. These architectural re-design measures would increase the difficulty in building large DDoS attack networks. Whilst the proposed solution could prove unpopular to some (who view it as unacceptable management of the Internet and detrimental to the Internet’s intended flexibility), it is argued that problems associated with DDoS attacks are viewed as so dire as to merit the significant changes proposed to the existing Internet architecture.

High-speed programmable hardware boxes, such as Attack Mitigator from Top Layer [204] can help protect a network’s perimeter from a range of DDoS flooding attacks with the goal to prevent intrusive traffic from bringing down a site. From independent reviews and industry experts, Attack Mitigator appears to have sustained a healthy reputation as a high-class intrusion prevention system that can cooperate well with other security components and precautionary measures (such as firewall, intrusion detection, virus scanner, and redundant clustering solutions) to provide state-of-the-art network protection [143, 200, 219].

Whilst there are no magic solutions to thwarting the DDoS threat, employing state-of-the-art technologies and keeping abreast of new DDoS attack patterns and best practice defensive measures will help mitigate the threat of a major disruption to MASHIn security services. David Dittrich’s DDoS web page [53] is an excellent, regularly updated, resource on all important matters related to DDoS education including (but not limited to) a history of DDoS attacks, analyses
and talks on DDoS attack tools, DDoS defensive measures, and vendors marketing products in the DDoS space.

10.6 Trusted Computing Initiatives

Since the turn of this century, there has been a lot of interest and much conjecture surrounding “trusted computing” initiatives, particularly those arising from Microsoft’s Next-Generation Secure Computing Base [132] and the Trusted Computing Group [203].

The Trusted Computing Group (TCG), formerly known as the Trusted Computing Platform Alliance (TCPA), is an initiative led by Intel Corporation, Microsoft, IBM, Hewlett-Packard, Sun Microsystems, Sony Corporation, and AMD who claim to “develop, define, and promote open standards for hardware-enabled trusted computing and security technologies, including hardware building blocks and software interfaces, across multiple platforms, peripherals, and devices”. Their hardware specifications are designed so that computing industry vendors can utilise them in products that protect and strengthen the computing platform against software-based attacks. In contrast, traditional (mainstream operating system) security approaches have been solely software-based, making them vulnerable to malicious attacks, virtual or physical theft, and loss [203].

Next-Generation Secure Computing Base (NGSCB), formerly known as Palladium, is Microsoft’s initiative for its series of Windows operating systems which - in its own words - “are (technologies) designed to help provide better system integrity, information security, and privacy, offering a foundation to help ensure that privacy- and security-sensitive hardware and software can interact with greater integrity” [132]. Microsoft’s NGSCB anticipates architectural changes at a high-level within four groups, all of which require new hardware to be added to today’s PCs. The NGSCB project specifies software changes that take advantage of the security benefits made available by a planned new PC hardware design, including those changes proposed by the TCG. These changes should enable [178]:

- **Trusted memory curtaining**: A strong, hardware-enforced memory isolation feature to prevent programs from being able to read or write one another’s memory.

- **Secure input and output**: Aims to address the threats posed by keyloggers and screen-grabbers, software used by snoops and intruders to spy on
computer users’ activities.

- **Sealed storage**: Aims to address the inability of a PC to securely store cryptographic keys.

- **Remote attestation**: The intention is to remotely detect malicious software changes in a computer’s configuration.

There are a lot of negative opinions about what these trusted computing initiatives will mean for the future of computing - principally concerning implications to our privacy, user choice in configuring their machines, and increased dangers of monopolistic software/hardware lock-in. Security experts including Ross Anderson [10] and advocates of free software including Richard Stallman [185] both have strong pessimistic outlooks on these trusted computing initiatives. Many of the concerns raised seem to have some merit, though perhaps the scepticism and fear they try to instil in the public is a little exaggerated in some respects. We do not wish to get into a large debate on the politics and merits of the TCG and NGSCB initiatives. We will briefly discuss some concerns with respect to the purported technical advances arising from the trusted computing initiatives, in particular that trusted computing is unsuitable for discretionary access control mainstream operating systems designed for the average public end-user.

We agree with Reid and Caelli’s [159] argument that the services of sealed storage and remote attestation can *not* operate in a secure and efficient manner on mainstream discretionary access control operating systems (for which these initiatives are being primarily targeted). Mainstream operating systems lack the capability of enforcing fine-grained access control, in particular they do not support mandatory access control (see Section 10.2) and generally only have two main types of users (a root/superuser with full privileges, and normal users). A classic example where this can be malevolently abused is by device drivers which must run with full privileges, but are not necessarily trustworthy pieces of software - even if they are digitally signed. This operating system design flaw undermines the secure fault “ring” levels available in mainstream processing chips from Intel’s x86 processor architecture design since the 286 chip.

Remote attestation, which broadly aims to allow “unauthorised” changes to software to be detected [178], is neither a manageable nor fair solution. Attestation is not manageable because the value resulting from a remote attestation calculation must always reflect the current configuration, but this would require
all other processes to be managed on an ongoing, infeasible basis [159]. Furthermore, the computer owner may be treated as just another attacker (in terms of remote attestation) because a change in the computer’s configuration may break the computer’s ability to interoperate with other networked computers (since they remain unaware of the semantics behind a non-aligning remote attestation result) [178]. However, if a remote attestation query can return more meaningful results\(^4\), this may be more useful in both public end-user computing, and in remote attestation challenges between MASHs and checking hosts. In the latter (MASHIn) case, coupling operating system mandatory access controls with trusted computing concepts (as long as the technology is verifiable) may offer a more secure, stable solution for building confidence in the safe state of MASHs and checking hosts.

### 10.7 Summary

In the first half of the thesis (i.e. Chapters 2-5), we presented why we believe that the sheer volume and complexity of security threats that are associated with the general Internet mobile agent paradigm model, as well as the mistrusting major stakeholder problem, presents a very strong case for the design of a new high-level approach incorporating neutral, effective TTPs to not only reduce security threats but also bring the two major stakeholders closer to a working relationship.

In the second half of the thesis (i.e. Chapters 6-9) we discussed our visionary solution of achieving a more secure, harmonious, and practical Internet mobile agent community. Our proposed solution, the Mobile Agent Secure Hub Infrastructure (MASHIn), effectively uses traditional distributed system TTPs (like RAs, CAs, and AAs) along with novel, unbiased mobile agent paradigm-specific TTPs (namely MASHs and checking hosts) to remove the concern of the major stakeholders that their next move may be a losing one. Moreover, the MASHIn TTPs work effectively together to enable a transparent means of bringing previously untrusting mobile agent stakeholder entities closer to a working relationship.

However, we have never claimed that the MASHIn concept is a complete, readily-deployable solution. Protection of MASHs and checking hosts is critical to the viability of the MASHIn concept, since mobile agent users and mobile agent platform owners entrust these entities with a lot of security functionality

\(^4\)“Semantic” remote attestation of virtual machines has been considered [78].
concerning deployed agents and their verified safe, correct execution.

In this chapter we discussed some precautionary measures which need to be considered in effectively protecting MASHs and checking hosts. In Section 10.1 we discussed distributed deployment, backup, and replication measures which will all be required to ensure continuity of MASHIn TTP security services. Secure operating system issues including mandatory access control provisions were discussed in Section 10.2. Protecting stakeholder messages to/from MASHIn TTPs was discussed in the context of virtual private network connections in Section 10.3, where we also presented a more concrete (but still fairly basic) foundation network architecture for the MASHIn. We briefly touched on the importance of in-depth firewall design for protecting the non-Internet facing MASHIn WAN Intranet in Section 10.4, and addressed the serious problem of distributed denial-of-service attacks in Section 10.5. Finally, in Section 10.6 we considered trusted computing initiatives, and if they may be partially valuable for protecting MASHs and checking hosts.

In the next chapter we provide the interested MASHIn reader with some implementation recommendations for MASHIn design and implementation, and present three mobile application domain scenarios to highlight the strategic benefits of the MASHIn macro-level approach to mobile agent security which should be factored into designing crafty, autonomous and secure mobile agent applications.
Chapter 11

Implementation
Recommendations

Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.

Rich Cook

In Chapters 7-9 we investigated and re-focused three traditionally studied micro-level aspects of mobile agent security\(^1\), within a macro-level context - that being the MASHIn. It is not feasible to design and implement a large-scale novel infrastructural project such as the MASHIn in a three year Doctoral research programme (especially considering at all times there was only one participating project investigator, that being the candidate\(^2\); indeed conceptualising and communicating the new ideas presented in this thesis has been a mammoth task in this time period. In this chapter we present the reader with a review of implementation work completed, specifically on SCC4MAP, and recommendations for

\(^1\)Specifically mobile agent (code and data) privacy, execution integrity, and agent reputation respectively.

\(^2\)Even with the hypothetical assistance of many co-workers, designing and implementing the MASHIn would take longer than three years.
proceeding with further MASHIn design and implementation. Wherever possible we point the reader to existing mobile agent system research implementations or frameworks which can be utilised or should at least be considered by those proceeding to implement (or produce more detailed designs of) MASHIn components in the future. In Section 11.1 we start by revising the implementation work completed on our novel SCC4MAP mobile agent code and data privacy strategy, and provide the reader with some broad general technical issues for consideration when further implementing the MASHIn. We consider implementation considerations for our RECDAM execution integrity strategy in Section 11.2 and our confidence scores framework in Section 11.3. Scalability of the MASHIn is discussed in Section 11.4. In Section 11.5, we take a look at three mobile agent application case scenarios, and discuss patterns (and associated implications) for how they could be implemented in a MASHIn context compared to the more traditional generic micro-level agent platform only situation. The chapter is rounded-off with a summary in Section 11.6.

This chapter includes and builds on material found in peer-reviewed accepted papers [i]-[iv] listed on page xxi.

11.1 Technical Considerations

A lot of the PKI functionality and systems (e.g. VPN) integration in the MASHIn can be built on many years of prior real world experience and well established international de-facto standards [3, 145, 172, 216]. The old adage - do not re-invent the wheel - should be adhered to.

However, if one directly reuses existing components from other distributed architectures the MASHIn will not be immune to inheriting the good and the bad features from those existing distributed systems, especially with respect to PKIs. Where possible, future design of the MASHIn should try to learn from, and obviously try to avoid, limitations with existing PKIs (such as those problems discussed towards the end of Section 6.2).

The MASHIn, as a high-level infrastructural proposal, to promote interoperability of heterogeneous agents should - where both feasible and appropriate - adhere to leading mobile agent standards including the OMG’s Mobile Agent Facility (MAF) Specification [146] and those of the Foundation for Intelligent Physical Agents (FIPA) [62].
The MASHIn must be capable of supporting application and language independent mobile agents. This means that the MASHIn TTP entities directly interpreting mobile agent code, specifically MASHs and checking hosts, must be flexible in that they can support a suite of language interpreters. Even though the vast majority of today’s mobile agent systems are constructed for Java programmed agents to be interpreted on a JVM, there may be a demand for mobile agents constructed in other languages - particularly other popular interpreted languages such as Perl and Microsoft’s C# language.

Custom class-loaders for facilitating the MASHIn SCC4MAP’s ‘real’ and ‘dummy’ callback classes (see Section 7.3) for achieving mobile agent privacy will need to be implemented. We successfully built a prototype mobile agent application which demonstrated that such a customised class loader could be constructed for Java-based mobile agent systems. This prototype was specifically discussed in Section 7.3.3, and some details on class relationships including interfacing with the resulting custom class loader from our SCC4MAP proof-of-concept are provided in Appendix B.

Whilst there have been many mobile agent systems constructed [87], very few have substantial security models incorporated in their design and implementation. In Table 11.1 we provide a reference to five of the better mobile agent systems; all were (and some still actively remain) focused, at least with a significant concentration, on research and investigation of mobile agent security issues. Even though the MASHIn is an infrastructural-level (hence higher-level abstraction than the mobile agent system-level) security proposal, it can and should build on the experiences and expertise of other researchers in the field.

Out of the five listed mobile agent systems, SeMoA is the most comprehensive and actively supported - with no obvious disadvantages in investigating it for possible future utilisation of some of its sub-components. It appears that Mole and Gypsy have not been enhanced since 2000 and their associated projects have come to a completion, but nevertheless those project’s produced useful research outcomes which can be considered in implementation and future research. To consider one example - Fritz Hohl, the security architect for Mole, devised the reference states protocols that we enhanced and built-upon in producing the MASHIn’s RECDAM execution integrity scheme. We discuss this related work of Hohl’s in assisting with possible future implementations of RECDAM in the next section.
<table>
<thead>
<tr>
<th>Mobile Agent System</th>
<th>Produced By</th>
<th>Security/Research Strengths Include</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeMoA [64]</td>
<td>Fraunhofer Gesellschaft, Germany</td>
<td>Strong agent sandboxing, secure class loading, filtering of virulent code possible, agent data integrity interleaving protocol attack resistant, secure agent tracking and communication, RBAC, follows international security standards, design interoperable with other agent systems, modular service-oriented architecture, easily set-up and configurable, pipeline principles for reasonably simple extensions of security functionality, supports recent Java release bundles, ongoing development/documentation/support</td>
<td>–</td>
</tr>
<tr>
<td>SOMA [205]</td>
<td>Università di Bologna, Italy</td>
<td>Tools for constructing flexible security policies concerning agent interaction, multi-hops integrity protocol, prominent use of TTPs in security architecture, follows standards for interoperability, mobile agent resource usage monitor/controller, mobile agents used for QoS and network management purposes</td>
<td>Adhoc system re-design and development code change management</td>
</tr>
<tr>
<td>Ajanta [206]</td>
<td>University of Minnesota, USA</td>
<td>Network management security/robustness, policy-driven secure collaborative systems, secure agent migration/inter-messaging/access control/remote monitoring, URN-based entity naming and name resolution</td>
<td>Incompatible with newer Java releases, very little research focus on the malicious agent platform problem</td>
</tr>
<tr>
<td>Mole [208]</td>
<td>University of Stuttgart, Germany</td>
<td>Reference states for agent execution integrity, code mess-up algorithms and time-limited blackbox security, reliability and fault-tolerance of agents, rollback of mobile agent execution</td>
<td>R&amp;D discontinued</td>
</tr>
<tr>
<td>Gypsy [196]</td>
<td>Technical University of Vienna, Austria</td>
<td>Component-based mobile agent design, mobile agent design pattern definition, direct support for mobile agent supervisor-worker framework (discussed in Section 4.2.4.8) [61]</td>
<td>R&amp;D discontinued</td>
</tr>
</tbody>
</table>

Table 11.1: Mobile agent research systems with strong security perspective.
11.2 Execution Checking

In one (that being [86]) of his two reference states specific research papers (the other being [85]), Hohl provides some useful hints on how to implement reference state functionality in an agent’s mission for execution checking purposes. Specifically in Section 5 of that paper [86], Hohl provides two callback methods in the agent and five interfaces implemented by the agent which we will now discuss.

Callback method `checkAfterSession()` is the first action taken by the next agent platform - i.e. the execution checking first step on the next agent platform (refer to Figure 8.2 on page 173) in an agent’s mission. This next agent platform can easily collude with the previous agent platform to hide malicious alterations to the agent’s code or state - one of two major problems with Hohl’s scheme (the other being inputs from agent platforms cannot be kept secret from each other).³

In the MASHIn’s RECDAM execution integrity scheme, discussed in detail in Section 8.3.2, we detail an agent’s itinerary restructuring and changes to the reference states protocols which make Hohl’s underlying reference states scheme much more robust. The execution checks in RECDAM are performed on neutral TTP checking hosts (not on potentially competitive and/or maliciously colluding agent platforms), which solves the two major problems with Hohl’s scheme and helps to assist in solving a number of other minor problems.

Hohl gives the agent programmer the option of codifying what reference states checking functionality he/she wants in the `checkAfterSession()` callback method. In an implementation of the MASHIn’s RECDAM scheme, this could be supported. However, recall execution integrity checking is important from both major stakeholders perspective - obviously the agent user does not want his/her agent’s execution integrity manipulated with, but agent platform owners also want to be irrefutably cleared of any suspicion in attacks against an agent’s execution integrity. Thus, the checking algorithm must be generically defined in the MASHIn and encoded in checking hosts (who are supplied with a reliable previous agent input state from the previous checking host or MASH, and non-repudiable agent platform inputs and a resulting agent state from the agent being interpreted on that agent platform). A generic, non-conditional callback will therefore be interpreted by the checking host on every agent passing through. This is not to say

³See Section 8.3.1.5 for a discussion of limitations we identified with Hohl’s reference states execution integrity scheme.
that an agent user-defined checking callback method cannot additionally be sup-
ported - such code could be encapsulated in the afterEveryHost() callback
method suggested for the MASHIn’s SCC4MAP mobile agent privacy scheme
(see Section 7.3.2 for more details). Following on from this, agent user-defined
code encapsulated in Hohl’s checkAfterTask() callback method could be en-
capsulated in the MASHIn SCC4MAP’s afterMission() callback method,
interpreted by the MASH4.

For similar reasons to those just explained in discussing Hohl’s two reference
states agent callback methods, Hohl’s five interfaces implemented in agents are
not necessary within a MASHIn RECDAM implementation. Consider the first
two interfaces described by Hohl: InitialStateRequestor and Resulting-
StateRequestor - these two inputs/outputs must be explicitly supplied as part
of the RECDAM protocols in relation to checking hosts and agent platforms re-
spectively. Once again, however, agent user-defined interface implementations
can be supplied additionally. It is important to note that these really serve to
favour the agent user, whereas the RECDAM protocols and objectives (and in-
deed those of the MASHIn generally) are to fairly support the security interests
of both major stakeholders - this is why RECDAM prescribes mandatory inputs
and outputs from the respective stakeholders, irrefutably verified and maintained
by checking hosts and MASHs. Containers for packaging, and lower-level mech-
аниsms for working with, the required RECDAM inputs and outputs from both
major stakeholders and auxiliary TTPs involved in the execution checking pro-
cess (i.e. checking hosts and the MASH) must be defined and integrated into
a working, transparent solution - possibly utilising a modified virtual machine
environment, as suggested by Hohl [86].

Though Wilhelm et al. [229, 225] acknowledge that their ideas are purely con-
ceptual and are yet to be implemented, they rely on trusted tamper-proof agent
platform environments to ground a pessimistic view of trust on agent platform
ethical behaviour - but, with the caveat that constructing such a device to ef-
effectively resist tampering is a decidedly nontrivial task and logistically would
be hard-pressed to attract wide agent platform vendor uptake. A more realistic
scenario, suggested in the previous chapter, is to only rely on these so called
Trusted Processing Environments (TPEs) to be installed in MASHs and check-
ing hosts (not agent platforms). Many of the limitations recognised by Wil-

4This is a strong MASHIn advantage over traditional approaches in that it keeps the agent’s
mission completion totally autonomous of its agent user.
11.3 Confidence Scores

Implementing a practical confidence score framework for the MASHIn, specifically the novel framework proposed in Chapter 9 will not be a simple task, even though stakeholder confidence scores calculated from MASHIn TTP first-hand observations are more reliable than those based on second-hand opinions (the source type from which most reputation data is collected in P2P systems and social control agent frameworks) [124, 25, 2, 156]. The real difficulty, from a MASHIn confidence score implementation perspective, arises from three major factors:

- The MASHIn’s principal reputation framework is centralised (i.e. first-hand observation reliant) - based on MASHs and other reliable MASHIn TTP first-hand observers such as checking hosts. Effectively prototyping the dynamics of a sizeable community (that the MASHIn is aimed at) will require potentially hundreds of agent platforms\(^5\), a larger number of agent users deploying - ideally not canned - mobile agents, and a proportionally reasonable number of MASH/checking host sources providing the feedback. This will likely require research cooperation across institutional centres and ongoing unguided user interest.

- Implementing, or acquiring and integrating, appropriate sensors for sourcing useful confidence score raw data - such as virus scanners, and agent logic

\(^5\)Certainly, in the dozens.
bomb filters - will likely be expensive and time-consuming. Ironically, the more reliable the sources of reputation - the less information that is collected on peers\textsuperscript{6}.

- Implementations of RECDAM and potentially other execution integrity protocols will be needed, as well as designing secure and efficient protocols to update a central repository\textsuperscript{7} from which MASHs can analyse collected confidence score data and formulate personalised, private confidence scores for both major stakeholder sets.

Research on the SeMoA mobile agent system [64] has produced some highly customisable in/out gateway filters for transparently identifying and limiting the propagation of agent logic attacks (such as some simple denial-of-service threats). It would be the best agent system we know to utilise in the MASHIn confidence scores prototype for not only this reason, but also due to the fact that the SeMoA team have commenced the co-ordination of a global, cross-institutional SeMoA mobile agent network with systems located in three continents - Europe, North America, and Asia.

Before a credible MASHIn confidence score prototype can commence the infrastructure and logistics suggested above will need to be in place, and supporting confidence score criteria and protocols/formats for transporting/storing confidence score data need to be designed.

### 11.4 Scalability of the MASHIn

The MASHIn, in peak practice, should be able to support thousands of agent platforms and millions of agents, though it is unlikely that handling this number of active agents at any one time would be required often or at all. Whilst these sound like quite large numbers, facilitating an appropriate level of support would not be as daunting as one might initially think, for a number of reasons including:

\textsuperscript{6}The primary sources for a peer’s reputation (in P2P reputation systems) have been grouped as: personal experience, external trusted sources, one-hop trusted peers, multi-hop trusted peers, and a global system. Each source provides increasingly more information about peers. However, that information becomes increasingly less credible as well [124].

\textsuperscript{7}Even though the confidence score data may in fact be stored within a distributed database, the storage abstraction remains centralised from the perspective of the framework and the major stakeholders.
• Additional traditional TTPs required in the MASHIn - for example registration authorities, certification authorities, and attribute authorities - can be added into the infrastructure on a need-by-need basis. Their commissioned services are remunerated on a standard fee (per term or transaction load as appropriate), as in other existing distributed systems.

• MASHIn-specific new TTPs - for example MASHs and checking hosts - can also be added into the infrastructure on a need-by-need basis. Their commissioned services are remunerated on a direct usage basis. We discussed possible MASHIn-specific remuneration models in relation to supporting mobile agent execution integrity checking and formulating entity confidence scores on behalf of the major stakeholders (respectively in Sections 8.3.3.4 and 9.3.4).

• These traditional TTPs and MASHIn-specific new TTPs are distributed across the infrastructure, assisting with load and congestion management (as well as minimising the transport penalties of incorporating TTPs into an agent’s itinerary).

• Secondary storage media is increasingly very inexpensive, and other required hardware such as load balancing components (a requirement in the MASHIn outer firewall/DMZ area) can be upgraded and financed on the increased number of MASHIn user subscriptions.

• The Internet, the underlying “playing field” of the MASHIn, is a network multitude times greater and busier than the MASHIn would ever require. The Internet scales well and, in most cases, inherently supports respectable procedures for dealing with potentially problematic scenarios such as mass network flooding (possibly from a series of propagating Internet worms), broken network routes, and unresponsive domain name servers.

In addition to the above-listed factors which promote ease of MASHIn scalability, the underlying novel (re-focused micro-level) schemes (to a macro-level perspective) of the MASHIn approximately double the overhead compared to non-secured mobile agent scenarios. This is because of the extra computation (for example, interpretation of ‘real’ callback classes in terms of mobile agent privacy in the MASHIn’s SCC4MAP scheme and the execution checking step in terms of
mobile agent execution integrity in the MASHIn’s RECDAM scheme - both performed on checking hosts or the MASH). One could also factor in the extra costs of transport to checking hosts, but this is necessary if any type of irrefutable real-time security and localisation of attack damage is possible between the two major (mistrusting by default) stakeholders. Overall, we think the overhead is not too costly given the significant security benefits of the macro-level MASHIn approach; furthermore, we see no reason why these MASHIn overheads should be anything other than very manageable. However, as discussed in Section 10.5, a real threat to MASHIn provision of security services - and in fact any online real-time system - could be faced from distributed denial-of-service (DDoS) attacks. It is paramount that future design work on the MASHIn strongly considers best practice measures for mitigating DDoS threats.

11.5 Application Case Scenarios

In this section we consider three mobile agent application case scenarios, and discuss how the MASHIn model helps solve or at least ease some of the security problems with the more general (non macro-level TTP) mobile agent paradigm alternative. We also try to highlight an example of how the MASHIn’s mobile agent privacy, execution integrity, and confidence scores can work respectively in the following three scenarios: a purchase shopping mobile agent scenario in Section 11.5.1, a group meeting negotiator scenario in Section 11.5.2, and a fairer research paper reviewer scenario in Section 11.5.3.

11.5.1 Purchase Shopping Mobile Agents

In Section 7.3.3 we discussed a purchase shopping mobile agent which visited an arbitrary number of agent platforms searching for some item (or set of items). In constructing the agent, it was dictated that the agent should immediately initiate a purchase (via the afterEveryHost() callback method) once an agent-user defined bargain price has been found, or alternatively purchase the best vendor offer price (via the afterMission() callback method) if a bargain price has not been found (and all vendor agent platforms have been visited) and the best vendor

---

8Recall that this overhead can be minimised by MASHs choosing checking hosts in the network vicinity of the related agent platforms.

9The MASH can act as a proxy-purchaser if the agent user does not trust the MASH with its payment details.
offer price is at least as good as an agent-user defined acceptable price. Using MASHIn SCC4MAP ‘dummy’ callback classes, the agent-user defined bargain and acceptable prices are not disclosed to the vendor agent platforms (these reside in the ‘real’ callback classes of the agent - privy only to the MASH and checking hosts incorporated in the agent’s itinerary by the MASH). This is an example of mobile agent data privacy in SCC4MAP. An example of mobile agent code privacy in SCC4MAP is the logic to purchase the item/items when a bargain is found, or alternatively purchase the best priced item (if at least as good as the acceptable price), or if no acceptable price is found perhaps query other vendor agent platforms or search for other similar item/s or proceed no further.

Neither this data nor code autonomous mobile agent privacy is possible in the traditional mobile agent paradigm\(^\text{10}\) due to the fact that an agent platform in the general model must have access to the code/data of an agent for it to be interpreted. The MASHIn, thus, provides explicit infrastructural support (via MASHs and checking hosts) to enable mobile agent data and code privacy in a macro-level context, not favouring one or the other of the major stakeholders\(^\text{11}\).

### 11.5.2 Group Meeting Negotiator

Consider a scenario whereby a team leader wishes to re-schedule a meeting (for example, due to illness), with the conditions that the meeting be attended by everyone of the intended employees (potentially an arbitrary number, but for this scenario say four people) at the earliest common-time available to every intended employee; if this is not possible in the next two days, the earliest time in the next two days which is convenient to most of the intended employees should be scheduled. Though this problem could, in general, be solved directly by employing only client-server technology it is better suited (in this case) to employing mobile agent technology because no central database of employee schedules is maintained and each of the employees works on location at customer premises - thus, connectiv-

\(^{10}\) Though, it is possible to attain similar data/code privacy by employing either the supervisor-worker framework [61] (see Section 4.2.4.8) or mutual protection of cooperating mobile agents [164] (see Section 4.2.4.1) approaches. The big drawback with these approaches is that if either a trusted platform (for the supervisor) or set of trusted agent platforms (for the cooperating agent) cannot be ascertained - quite a likely scenario in the Internet - then those schemes fall down (reverting to the home agent platform, for example, for security) and break the autonomous property of mobile agents.

\(^{11}\) We also discussed how agent platform input values (into the agent) can be kept secret from other agent platforms in Section 7.3.5.
ity is sometimes limited by mobile device connection black-spots\textsuperscript{12} and employees may be in the middle of a job for a customer and cannot be interrupted.

A mobile agent’s itinerary is set such that the agent visits each employee. On mobile agent arrival (and when first convenient to the employee), the first employee enters his/her available times in the next two days. The mobile agents proceed to the second employee (in the itinerary) who enters his/her available times in the next two days, and so on to the final employee. The mobile agent, during its mission, maintains a list of common-times available to the employees who have entered their available times and should present the next employee with the common-times first (to relieve employee/employer calculation of these common-times).

In the above scenario, one can imagine threats to the mobile agent coming from a disgruntled employee who may, for example, refuse to pass on the mobile agent to the next employee or purposely manipulate the agent to claim a false common-time list. In the general mobile agent model, neither of these threats can be detected/mitigated because they involve threats directly to the mobile agent’s autonomous dynamic movement (in the first case) or the mobile agent’s dynamic data (in the second case). However, in the MASHIn, a MASH (along with cooperation from its delegated checking hosts incorporated into the agent’s itinerary) can track agent TTLs to mitigate the first threat; MASHIn agent TTL monitoring was discussed in Section 6.5. To mitigate the second threat, the MASHIn’s RECDAM scheme (presented in Section 8.3.2) will irrefutably tie each employee’s available times to their inputs and also be able to detect execution integrity changes to the agent’s dynamic data (via the underlying reference states checks co-ordinated by trusted checking hosts).

The optimisation of common-times can be done safely via SCC4MAP secure callback methods forwarded to the checking hosts, and the final best time for the employees group meeting with employer can be calculated safely and autonomously on the MASH at itinerary completion. If the agent has been directed to autonomously book the best time, then it can proceed to autonomously inform each of the employees of the common meeting time (or perhaps, if every employee cannot make that time, provide him/her with alternative informational arrangements).

One can also imagine scheduling mobile agent application scenarios whereby

\textsuperscript{12}Areas in which mobile devices, such as mobile phones and PDAs, cannot access the telecommunications network and/or Internet.
employees holding different superiority positions could have different access right levels to query and/or update fellow employee schedules. This is an example application in which utilisation of the MASHIn’s credential infrastructure framework with roles and rights (encapsulated in attribute certificates that travel with the mobile agent) would be practical; the MASHIn’s credential infrastructure framework is discussed in Section 6.3.3.

11.5.3 Fairer Research Paper Reviewer

One can imagine many applications of mobile agents within the research domain, assisting varying principal groups - including researchers, paper reviewers, and conference organisers - involved in the research process. In Section 8.1.1 an example mobile agent application within the research domain was outlined, specifically one in which quality acclaimed authors or papers could be assessed and tracked down based on a weighted-average of collected ratings from well-renowned research principals. In the ensuing discussion below, we will outline another mobile agent application within the research domain, specifically one which could promote a higher quality-assured research paper reviewing process.

The idea is to irrefutably produce a historical record of reviewer results, assessing factors such as positive/negative standard-deviations of reviewer results compared to fellow reviewers. To take an example, international peer-reviewed research conferences with a reputation of attracting high quality research papers and coordinated by a respectable organising committee generally prescribe that each submitted research paper must be reviewed by at least three research professionals expert in the field relating to the specific research paper under review. Consider the review of one paper for a moment; if the paper receives a rating of 4 and 5 (out of 10) from two reviewers and an 8 or 9 from another reviewer, then there is reasonable grounds for probable concern with the significantly higher result. The deviation (in this case a positive deviation for the third - highest scoring - reviewer) is not negligible. Thus, if an independent third party (or a series of co-ordinating mobile agents based at independent trusted third-party agent platforms) could collate a historical record of numerous statistics\textsuperscript{13} it could lead to a better level of reviewer credibility and, at a minimum, offer reviewers a

\textsuperscript{13}These statistics could include, but do not have to be limited to: which reviewer reviewed which papers, having which paper authors (with tracked affiliations - e.g. continent, country and institution), reviewer average scores and average result standard deviation with respect of other reviewers.
simple method for self-critique.

The central independent third-party (potentially a stationary agent on a MASH) could communicate (via secure inter-agent messaging) with mobile agents it sends around to organising institution agent platforms to collate, store, and analyse reviewer statistics. The reviewer-specific results/statistics could be kept private from other reviewers by strict access control rules for querying the independent third-party. As mentioned, each individual reviewer should be able to query his/her historical performance\(^{14}\) for self-evaluation purposes. Conference organising committees, when deciding on high-quality and equitable reviewers, can query the independent third-party for eligible (and available) reviewers meeting specific criteria and research expertise. Other usage of mobile agents within this refined domain are possible - for example automated dispatching of conflict-resolution mobile agents to search for another parties’ opinion on a deadlocked paper review case, reducing duplicate submissions of papers not encapsulating significant new work to more than one conference or journal (and potentially penalising the offending author/s), or mobile agents assisting paper authors in the search for conferences with historically fair reviewing statistics. All involved parties (paper authors, reviewers, conference organising committee, and independent third-party sources analysing results) would register and receive valid MASHIn public key-pairs, as well as attribute certificates encapsulating their roles and rights within the reviewing system.

The MASHIn, given the above review result system outline, would be able to support and publish personalised, private stakeholder confidence scores based on first-hand observations (of reviewer digitally-signed commitments of their paper review results) and/or support second-hand opinions (from organising committees) on the neutrality and quality of reviewer performance\(^{15}\). A novel framework for constructing macro-level mobile agent stakeholder confidence scores, in the context of the MASHIn, was presented in Chapter 9.

It should be clear that without a significant infrastructural abstraction, potential mobile agent applications of the size and flexibility outlined in this subsection will remain infeasible - and certainly not achievable in an unbiased, irrefutable

\(^{14}\)If deemed in violation or contradiction of governing privacy laws or ethics, the deviation to specific-named principal reviewers can be kept private by the TTP MASH.

\(^{15}\)Besides final paper score deviations, reviewer performance could be expressed with respect to many other attributes including reviewer timeliness and other qualities such as whether their reviews are understandable, concise/detailed/verbose, and/or provide helpful feedback to the committee and authors.
manner. The MASHIn is admirable because it is the first major security infrastructure in terms of mobile agent networks, that we are aware of, which provides macro-level unbiased major stakeholder security support.

11.6 Summary

This chapter served two main purposes. Firstly, the chapter presented the reader and potential interested MASHIn researcher and/or developer with some guidance on issues to consider when pursuing further design and/or implementation of the MASHIn. Relevant topics addressed here included technical MASHIn issues and expanding on the base MASHIn high-level design, scalability and overhead of the MASHIn, and the micro-level re-factored solutions (in a macro-level context) of mobile agent data and code privacy (i.e. SCC4MAP), mobile agent execution integrity (i.e. RECDAM), and the MASHIn framework devised for constructing reliable confidence scores on major stakeholder behaviour. Secondly, the chapter presented a series of three different mobile agent application domain scenarios which helped to highlight why the macro-level framework of the MASHIn can be advantageously indispensable for one wanting to design crafty, safe and autonomous mobile agents on the Internet.

In the next (final) chapter we review the contributions of this thesis to the existing body of knowledge. We also present our conclusions from undertaking this research, and discuss possible avenues for future work associated with the MASHIn concept as a framework for deploying safe and autonomous mobile agents on the Internet.
Chapter 12

Conclusions and Avenues for Future Work

Travel is only glamorous in retrospect.  

Paul Theroux

Since the lofty marketing expectations of its early years, interest in the mobile agent paradigm has wavered but generally slowed due, in large part, to non-trivial security problems related to its processing model. However, as we as a society become ever-increasingly pressed for time, flexibility for better processing larger banks of information and completing tasks more efficiently will be sought. The approach of delegating tasks to mobile agents, so that we can work on other tasks concurrently while the mobile agent works to complete some task mission on our behalf is an attractive prospect. Mobile agents will be most useful if they are autonomous entities, i.e. they do not seek interaction with their user (who may be off-line due to mobile device limitations/costs or pre-occupied with some other task) during their mission undertaking.

Nevertheless, for this flexibility to become widely practical, mobile agent services must be ubiquitous and a standard framework for non-maliciously processing mobile agents is needed. A way of assuring safe outcomes for both major stakeholders - mobile agent users and mobile agent platform owners - must be found. This thesis takes the position that, in seeking safe outcomes for the major
stakeholders, the security concerns of both stakeholders should be studied and addressed together without bias towards either stakeholder. In this endeavour, a high-level infrastructural strategy was devised to assist in bringing the traditionally mistrusting major stakeholders closer to a working relationship in an equitable environment. At the same time, direction towards a standard framework for safely deploying application and language-independent autonomous mobile agents was provided.

12.1 Contributions

A major outcome of this thesis was the conceptualisation and foundation design of the macro-level Mobile Agent Secure Hub Infrastructure (MASHIn). The MASHIn is a novel strategic proposal for potentially supporting and uniting millions of agent users and thousands of agent platform owners in a safe, accountable fashion. The MASHIn is a macro-level conception because it addresses, without bias, security issues of concern to both stakeholders. Besides an introduction to the MASHIn, the thesis analysed a lot of background research material on mobile agent security matters, and proposed some new schemes for achieving desired security objectives for the major stakeholders - however, in contrast to the existing schemes, the new schemes were designed in the context of a macro-level perspective.

Following is an account, in point form, of some of the contributions made to the mobile agent security field in producing this thesis (where possible in order of presentation):

- A non-security-specific background primer to the mobile agent field including what constitutes a mobile agent and its state life-cycle, where the mobile agent paradigm has derived from, applications naturally geared towards utilising the mobile agent model, and general factors limiting ubiquitous mobile agents.

- Providing an understanding of why protecting the security interests of stakeholders in the mobile agent paradigm is particularly more difficult than protecting the security interests of stakeholders in more traditional distributed system paradigms.

- A non-exhaustive, but nevertheless accurate portrayal (and discussion) of
many security threats which are tied to the mobile agent paradigm. Lists were constructed specific to both paradigm major stakeholders - mobile agent platform owners and mobile agent users.

- A non-exhaustive, but nevertheless accurate portrayal of a good sample of the state-of-the-art countermeasures to mitigating the previously listed threats. A summary account on the threat mitigation coverage of the state-of-the-art countermeasures highlighted a general trait that researchers have approached tackling mobile agent security issues from a micro-level perspective (i.e. focus on protecting the interests of only one of the two major stakeholders, and even then often only mitigating one or two threats). The micro-level focus limits interoperability of countermeasure approaches, and does not address the significant limitation of mistrusting major stakeholders.

- A review of existing schemes which have utilised TTPs in the mobile agent security address-space, and commented on how they are usually afflicted by either favouring protection of one of the two paradigm major stakeholders, or tied to application-specific mobile agents.

- Proposed a number of mobile agent (infrastructural-level) generic TTP security services, and explained how they can assist both major stakeholders.

- An overview of the non-MASHIn (i.e. traditional distributed system) TTPs and MASHIn-specific novel TTPs (namely MASHs and checking hosts), and how they may interact to provide the foundation for an infrastructural solution accommodating the security interests of both major stakeholders - conceptualising the MASHIn, its objectives, expected core components, and deployment considerations.

- Discussion on how employing RBAC and Jansen’s privilege management scheme for mobile agents can provide a flexible access control management solution to agent platform resources in the MASHIn.

- Discussion on how the MASHIn design permits flexible solutions to entity authentication, anonymisation of agent users (and, possibly agent platforms - and later, MASHs and checking hosts), agent location and itinerary event milestone monitoring, and fault-tolerant mission completion.
• An extended review of the state-of-the-art existing mobile agent data and code privacy mechanisms, and a comparative analysis with the newly devised Secure Callback Classes for Mobile Agent Privacy (SCC4MAP) mechanism (along with a discussion on a successful proof-of-concept implementation). SCC4MAP rated favourably against the existing mobile agent privacy schemes.

• Definition of new criteria for assessing the robustness of mobile agent execution integrity schemes.

• An extended review of the state-of-the-art existing mobile agent execution integrity mechanisms, and a comparative analysis with the novel Real-time Execution Checking and Deterrent Against Misbehaviour (RECDAM) scheme. The new criteria (mentioned in the previous bullet point) were used in this comparison, with the RECDAM scheme proving favourable in terms of robustness in meeting our desired properties for a solid execution integrity scheme. In the process of designing RECDAM, many improvements were made on Hohl’s existing reference states scheme; Hohl’s scheme was critiqued in greater detail than the other existing schemes as it has some particularly admirable objectives that were of special interest.

• Presented the first framework, that we are aware of, for building confidence scores (i.e. reputations) on both major stakeholders in a mobile agent network. MASHIn confidence scores were shown to be unbiased and more reliable over traditional second-hand opinions - and can be strategically utilised to transparently bring mistrusting stakeholders to a working relationship (based on private, personalised criteria and weighting scores passed by a requesting party onto a MASH for calculation).

• The new mobile agent privacy, mobile agent execution integrity, and agent (stakeholder) confidence score schemes were studied in a macro-level context (i.e. the MASHIn framework), and it was concluded that they were more reliable and accommodating of both major stakeholder needs than existing micro-level studied alternatives.

• Preliminary investigation into technologies and research areas of interest for utilisation in protecting MASHs and checking hosts (such as DDoS countermeasures and mandatory access control), as well as supplying a basic
12.2 Avenues for Future Work

Whilst a lot of work was covered in producing the MASHIn concept as presented in this thesis, there are many interesting avenues for future research investigation arising from this novel framework. Only a few possibilities are mentioned here; thereafter the reader’s ingenuity is left to run wild.

Reasonably simple fault-tolerance mechanisms for mobile agents in the MASHIn were discussed, but of a higher criticality are fault-tolerant mechanisms supporting the continuity of security services provided by MASHs and checking hosts.

The major component modules suggested for MASHs (in Section 6.1.3) should be sub-divided further, and collaborative interfaces defined. MASH delegated entities should be abstracted and contractual responsibilities drawn-up.

Mechanisms for assuring MASHs and checking hosts function correctly need to be investigated. Building on Wilhelm’s idea of TPEs [227] (discussed in Sec-
tation 5.1.3) is one possibility\(^1\), semantic remote attestation (touched on in Section 10.2) coupled with mandatory access control primitives is another possibility, and DDoS precautions will need to be carefully considered. Perhaps one area which can, and probably should, be investigated is a number of independent smaller MASHIn networks - which may help mitigate DDoS attacks, as well as offering more alternatives in terms of models of trust to invest in for major stakeholder subscribers.

Secure protocols for transporting ‘real’ callback classes between MASHs and checking hosts, and appropriately loading/unloading the classes should be added to the SCC4MAP mobile agent privacy mechanism.

New execution integrity protocols can and probably should be supported in the MASHIn framework, to complement the underlying reference states processing model in RECDAM - which may not be sufficiently efficient in some mobile agent applications, for example where large amounts of input data need to be carried with the agent (for checking purposes). Whilst the security limitations in Hohl’s reference states scheme [86, 85] were largely overcome, the underlying efficiency drawbacks of the processing model are inherited.

A rich set of stakeholder behavioural criteria for calculating MASHIn stakeholder confidence scores should be investigated. Then a prototype network incorporating appropriate sensors for analysing the load and dynamic reaction from entities, given calculated MASHIn confidence scores, can be studied. The impact of adding second-hand opinions can also be studied. The effectiveness of dynamic, private, personalised calculated confidence scores in the MASHIn can be contrasted with other models of reputation, such as those applied in peer-to-peer networking.

A policy schema (and then a front-end translation tool) should be developed for expressing (and entering) mission protection guidelines from the agent user such as what to do with an agent once its mission is done (e.g. by default transport back to user), specify agent TTLs, access rules for inter-agent messaging, etc. *Ponder* appears to encapsulate a very good vocabulary, with accompanying tools, for expressing and enforcing policy in distributed systems [46, 180, 47, 55].

In terms of key management issues, how do the MASHIn-specific components interface with traditional registration and certification authorities to handle life-

---

\(^1\) Whereas Wilhelm was advocating TPEs for agent platforms, a better idea is to consider them for parts of MASHs and checking hosts - a more appropriate scenario given the cut-down functionality set that comprises TPEs.
cycle events such as entity key updates and revocations - for example, they need to be propagated to the external router so that agent user VPN connections are handled according to the correct current cryptographic key state.

As presented in the basic network architecture (Section 10.3), is the proposed WAN MASHIn Intranet the best choice? What other alternatives could offer a cheaper, more flexible, and secure solution?

In terms of areas or issues raised that were deemed out of the scope of the thesis’ purpose, the following could fall into future work (with the appropriate section where each was noted following in brackets):

- When no pre-determined agent itinerary is supplied, there will be a need for formulated query syntax (and semantics) for determining which agent platforms have appropriate services aligning with the agent’s mission - via a yellow pages-like directory service (Section 6.4).

- Criteria that MASHs will utilise in selecting the most appropriate checking hosts to insert in an agent’s itinerary, and algorithms for monitoring checking host and agent platform processing loads/response times (Section 6.5).

- Sharing and secret threshold schemes for distributing trust amongst MASHIn TTPs (Section 7.3.5).

- Investigation of reputation systems from other domains including P2P and e-commerce contract systems to build upon the framework presented for formulating confidence scores in the MASHIn (Section 9.1).

- Assuming the proposed WAN MASHIn Intranet network architecture is deemed the best course of direction to take - WAN hardware, routing methods, and protocols for managing WANs will need to be thoroughly investigated and carefully considered (Section 10.3).

That was actually more than a few ideas for future work avenues, but it is by no means an exhaustive list. The MASHIn is certainly a unique high-level proposal and a novel macro-level perspective for considering mobile agent security issues, so there are bound to be many different angles from which future work can be seen.
Chapter 12. Conclusions and Avenues for Future Work
Appendix A

Glossary

**AA**: “Attribute Authority”, an entity that is trusted by at least two entities to create and assign attribute certificates. More specifically, an authority trusted by the verifier (some entity to which domain access is targeted, for example an agent platform owner) to delegate privilege to some requesting entity (for example an agent user) who wants access to resources in that target domain [7]. It is important to note that the AA is responsible for the attribute certificates during their whole lifetime, not just for issuing them [207].

**AP**: “Agent Platform”, a networked host offering authorised agents an environment for performing agent work on.

**APO**: “Agent Platform Owner”, the major stakeholder with administrative control of an agent platform.

**Access Control**: Any mechanism by which a system grants or revokes the right to access some data, or perform some action [123].

**Agent Platform**: See AP.

**Agent Platform Collaboration Attack**: Two or more maliciously-acting, not necessarily consecutive, agent platforms in a mobile agent’s itinerary collaborate to undermine the agent’s safe mission completion or to cover-up their own un-
ethical behaviour.

**Agent Platform Owner:** See *APO*.

**Application-independent:** Appropriate for every subset of known or future mobile agent applications.

**Asymmetric Cryptography:** Asymmetric cryptography is cryptography in which a pair of keys is used to encrypt and decrypt a message so that it arrives securely. Initially, a network user requests a public and private key pair. A user who wants to send an encrypted message can get the intended recipient’s public key from a public administrator. When the recipient gets the message, they decrypt it with their private key, which no one else should have access to. This process is known as a *public key infrastructure (PKI)* [197].

**Attribute Authority:** See *AA*.

**Attribute Certificate:** A set of attributes and a public key certificate identifier that are made unforgeable by use of the digital signature created with a private key (usually of an *Attribute Authority*) [7]. The attributes can encapsulate authorised privileges, for example.

**Authentication:** The process of identifying an individual or some computer - often based on a username and password, though public-key crypto-based authentication is also common. Authentication merely ensures that the individual or computer is who he/she/it claims to be, but says nothing about the access rights of that entity [108].

**Authorisation:** The act of giving authority or legal power. Whereas *authentication* is used to establish the identity of a party to a transaction, authorisation is used to determine what privileges that party will enjoy [144].

**Autonomous (Mobile Agent):** The *mobile agent* is able to make mission itinerary changes to best meet its mission goals, and importantly without interaction with its agent user once the agent has been deployed. In terms of security,
autonomous mobile agents do not rely on their agent user or home agent platform for ensuring its mission is completed securely.

**CA**: “Certification Authority”, a trusted third party whose purpose is to sign certificates for network entities it has authenticated using secure means. Other network entities can check the signature to verify that a CA has authenticated the bearer of a certificate [199].

**COD (Paradigm)**: “Code-on-Demand” Paradigm, a model of distributed processing allowing a client user to “pull” some software code from a distributed server and execute the code locally. Java applets are a typical utilisation of the COD paradigm.

**CS (Paradigm)**: “Client-Server” Paradigm, a common model of distributed computation in which software is split between server tasks and client tasks. A client sends requests to a server, according to some protocol, asking for information or action, and the server responds [51].

**Certification Authority**: See *CA*.

**Checking Host**: In the *MASHIn*, checking hosts are *MASH*-assigned entities (neutral to both major stakeholders) inserted between untrusted agent platforms in a mobile agent’s itinerary. They are important for completing a number of security responsibilities, including: monitoring agent TTLs, enabling agent itinerary anonymity, providing a trusted-base for interpreting mobile agent ‘real’ callback classes (in terms of supporting mobile agent data and code privacy), performing agent execution integrity checks, and providing feedback to MASHs for the calculation of confidence scores (on both major stakeholders) in the MASHIn.

**Client-Server Paradigm**: See *CS (Paradigm)*.

**Code Privacy (Mobile Agent)**: Keeping sensitive code logic (i.e. instructions for an agent or the agent’s ultimate goal) secret from untrusted agent platforms. In the MASHIn, this is effectively achieved by encapsulating this sensitive code logic in ‘real’ callback class/es within our *SCC4MAP* scheme - with this code
only accessible to non-attacking MASH and checking host entities.

**Code-on-Demand Paradigm**: See COD *(Paradigm).*

**Confidence Score**: A level of confidence that an entity will behave ethically, based on observations of their past behaviour. MASHIn confidence scores are flexible in that the requesting stakeholder can specify which criteria to build the confidence score on, and the observational data can be collected only (if desired) from highly-reliable unbiased entities such as MASHs and checking hosts.

**Confidentiality (Message)**: Assurance that message information is not disclosed to unauthorised persons, processes, or devices [7].

**DAC**: “Discretionary Access Control”, a means of restricting access to objects based on the identity and need-to-know of users and/or groups to which the object belongs. Controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (directly or indirectly) to any other subject [7].

**DDoS**: “Distributed Denial-of-Service”, on the Internet, a DDoS attack is one in which a multitude of compromised systems attack a single target, thereby causing denial of service for users of the targeted system. The flood of incoming messages to the target system essentially forces it to shut down, thereby denying service to the system to legitimate users [197].

**DMZ**: “De-militarised Zone”, a computer or small sub-network that sits between a trusted internal network, such as a corporate private *LAN*, and an untrusted external network, such as the public Internet [107].

**Data Privacy (Mobile Agent)**: Keeping sensitive agent data (either originating from before the agent was dispatched or gathered from agent platforms) secret from untrusted agent platforms. In the MASHIn, this is effectively achieved by encapsulating this sensitive data in ‘*real* callback class/es’ within our SCC4MAP scheme - with this data only accessible to non-attacking MASH and checking host entities.
Denial-of-Service: See DoS.


Discretionary Access Control: See DAC.

Distributed Denial-of-Service: See DDoS.

Distributed System: A collection of (often heterogeneous) computing resources whose distribution is transparent to the user so that the system appears as one local machine. This is in contrast to a network, where the user is aware that there are several machines, and their location, storage replication, load balancing and functionality is not transparent. Distributed systems usually use some kind of client-server organisation [51].

DoS: “Denial-of-Service”, a security intrusion which causes a system to be damaged, and where the damage is sufficient to disable at least one of the services offered by that system [123].

Dummy Callback Class/es: These are callback classes for mobile agents, accessible to untrusted entities such as untrusted agent platforms. Sensitive code and data is not encapsulated in these ‘dummy’ callback classes, but rather in ‘real’ callback classes only accessible to trusted entities (e.g. MASHs and checking hosts). ‘Real’ and ‘dummy’ callback classes are part of the MASHIn’s SCC4MAP scheme for mobile agent code and data privacy.

Execution Integrity: The property that mobile agents will be interpreted on an agent platform according to their program logic, and both major stakeholders will be irrefutably tied to their inputs into that session (i.e. an agent user with his/her code, and an agent platform supplying any data inputs to the mobile agent’s execution).

First-hand Observations: Personal observations of the state/event pertaining to a mobile agent, for example a virus scan of the agent. In the MASHIn, first-
hand observations from trusted entities such as MASHs and checking hosts are very reliable since these entities (and thus observations) are unbiased and non-malicious.

**IKE**: “Internet Key Exchange”, is an *IPsec* standard protocol used to ensure security for VPN negotiation and remote host or network access. Specified in IETF RFC2409, IKE defines an automatic means of negotiation and authentication for IPsec security associations (SA). SAs are security policies defined for communication between two or more entities; the relationship between the entities is represented by a key. The IKE protocol ensures security for SA communication without the pre-configuration that would otherwise be required [197].

**IPsec**: Short for “IP Security”, a set of protocols developed by the IETF to support secure exchange of packets at the IP layer. IPsec has been deployed widely to implement Virtual Private Networks (VPNs) [107].

**Integrity**: A condition in which data has not been altered or destroyed in an unauthorised manner [211].

**Intranet**: Any private network that uses some or all of the protocols of the Internet [7].

**JAR (File)**: “Java Archive” File, a compressed archive file containing Java class files, filename extension: “.jar”. The Java Development Kit contains a tool called “jar” for creating .jar files, similar to the standard Unix tar command. As well as archiving and compressing the Java class files, it also inserts a “manifest” file which can contain information about the class files, such as a digital signature. Combining class files into a single archive file makes it possible to download them in a single HTTP transaction. This, and the compression, speeds up execution of Java programs delivered via the Internet [51]. Any other type of file can also be encapsulated in a compressed form within a JAR file and be protected by a digital signature for message authenticity and integrity.

**Java Archive (File)**: See JAR (File).
LAN: “Local Area Network”, a data communications system that (a) lies within a limited spatial area, (b) has a specific user group, (c) has a specific topology, and (d) is not a public switched telecommunications network, but may be connected to one [7].

LDAP: “Lightweight Directory Access Protocol”, a series of IETF Internet-standard specifications for a lightweight version of the X.500 global directory service. Both X.500 and LDAP define the interactions between directory components to provide a global directory service. It is similar to the interaction between hypertext components on the Web [187].

Lightweight Directory Access Protocol: See LDAP.

Local Area Network: See LAN.

MA: “Mobile Agent”, a software program that executes and travels through heterogeneous computer networks (usually on behalf of a human user), performing computations and autonomously moving from host to host as required to fulfill goals on behalf of their users [221].

MA (Paradigm): “Mobile Agent” Paradigm, a distributed processing model whereby an agent with code logic (i.e. the “know-how” for some task) is owned by a client user and usually initially located on that user’s platform, but some of the required resources are located on remote agent platforms. Hence, the agent migrates to a remote agent platform carrying the know-how and possibly some intermediate results with itself (this migration can be arbitrarily iterative to different agent platforms). The MA paradigm is different from other mobile code paradigms in that the associated interactions involve the mobility of an existing computational component. In other words, while in REV and COD the focus is on the transfer of code between components, in the mobile agent paradigm a whole computational component, together with its state, the code it needs, and some resources required to perform the task, is moved to a remote site [214].

MAC: “Mandatory Access Control”, a system-enforced access control mechanism that uses clearances and sensitivity labels to enforce security policy. MAC asso-
iates the programs a user runs with the user’s session security level (clearance or sensitivity label) and permits access to information, programs, and devices at the same or lower level only. MAC also prevents users from writing to files at lower levels. MAC cannot be overridden without special authorisations or privileges \[187\].

Also “Message Authentication Code”, a bit string that is a function of both data (either plaintext or ciphertext) and a secret key, and that is attached to the data in order to allow data authentication \[7\].

**MASH**: “Mobile Agent Secure Hub”, an authoritative entity responsible for ensuring a mobile agent’s mission is completed safely. A MASH is the Lynch-pin neutral TTP in the MASHIn; the specific MASH delegated agent mission protection responsibility cannot be pre-selected by an agent user (only some guidelines for selection can be provided by the agent user) - promoting its unbiased relationship concerning both major stakeholders. MASHs co-ordinate the insertion of checking hosts into the agent’s itinerary in-between untrusted agent platforms and are responsible for a number of security services directly or indirectly, including agent user and itinerary anonimisation, mobile agent data and code privacy, agent TTLs, calculating reliable confidence scores for both major stakeholders, and the execution integrity of agents and irrefutably linking the major stakeholders to their inputs in an agent’s execution session.

**MASHIn**: “Mobile Agent Secure Hub Infrastructure”, the novel macro-level infrastructure presented in the second-half of this thesis for reducing the security risks for both major stakeholders and transparently bringing them closer to working relationships. Strategic utilisation of both traditional TTPs (e.g. RAs, CAs, and AAs) and novel TTP abstractions (specifically MASHs and checking hosts) is pivotal in the MASHIn meeting its objectives. The MASHIn is a new promising perspective for enabling autonomous and safe mobile agents on the Internet, and includes novel micro-level security perspectives with regard to agent robustness, flexible and scalable privilege management for agent access control to agent platform resources, mobile agent data and code privacy (see SCC4MAP), mobile agent execution integrity (see RECDAM), and formulating private and highly-reliable stakeholder confidence scores.
**MAU**: “Mobile Agent User”, the major stakeholder who deploys mobile agent/s to perform some task on his/her behalf.

**Major Stakeholders**: In terms of mobile agent systems, these are either *mobile agent users* or *mobile agent platform owners* - the two entities with the greatest vested interest in the mobile agent paradigm.

**Mandatory Access Control**: See *MAC*.

**Message Authentication Code**: See *MAC*.

**Mobile Agent**: See *MA*.

**Mobile Agent/Platform Impersonation Attack**: Either a mobile agent or a mobile agent platform masquerades as another party by faking the identity of a legitimate agent or agent platform.

**Mobile Agent Infrastructure**: An environment capable of inter-operably supporting multiple different agent systems, and a large number of agents and agent platforms.

**Mobile Agent Inter-Messaging**: Mobile agents, potentially located at different agent platforms, can send each other cooperative or informational messages. In the MASHIn, this can be facilitated more securely (than the general MA model) by having messages only be sent from agents when they are located on trusted platforms - for example MASHs and checking hosts. The messages can be protected against intercepted read attacks via asymmetric cryptography, also allowing the receiver to safely determine the message originator.

**Mobile Agent Itinerary**: A mobile agent’s planned journey of agent platforms to visit. Itineraries may be fixed (i.e. list of agent platforms to visit and their order of visitation known in advance of a mobile agent’s mission posting) or non-fixed (i.e. any variation on fixed itineraries).

**Mobile Agent Life-cycle**: The valid states (and transitions) of a mobile agent -
including created, executing, suspended, disposed, querying for services, migrating, or arrived at an agent platform (see Section 2.3 for more information).

**Mobile Agent Misroute Attack:** An attack whereby a mobile agent ends up at a destination it was not intended to by its own code logic. Ultimate responsibility for such an attack lies with the sending agent platform (i.e. that platform may misroute the agent, or a third-party in-between itself and the next agent platform may intercept the agent and misroute it - but until the agent arrives irrefutably at its next intended destination, the previous agent platform has responsibility for its travel safety).

**Mobile Agent Paradigm:** See MA (Paradigm).

**Mobile Agent Platform Owner:** See APO.

**Mobile Agent Secure Hub:** See MASH.

**Mobile Agent Secure Hub Infrastructure:** See MASHIn.

**Mobile Agent System:** The environment and resources for enabling mobile agent work. The most common environment/language for building mobile agent systems is the Java platform.

**Mobile Agent User:** See MAU.

**Non-repudiation:** 1) The capability, in security systems, that guarantees that a message or data can be proven to have originated from a specific person. 2) Assurance the sender of data is provided with proof of delivery and the recipient is provided with proof of the sender’s identity, so neither can later deny having processed the data [7].

**PKI:** “Public-Key Infrastructure”, a framework established to issue, maintain, and revoke public key certificates accommodating a variety of security technologies, including the use of software [7].
PMI: “Privilege Management Infrastructure”, a collection of Attribute Certificates (ACs), with their issuing Attribute Authority’s (AA’s), subjects, relying parties, and repositories, is referred to as a Privilege Management Infrastructure [207].

Private Key: A private key is a secret key, used in asymmetric encryption. It is mathematically associated with a public key, but is kept secret. This is one half of a matching key-pair [123].

Privilege Management Infrastructure: See PMI.

Public Key: A publicly distributed key, used in asymmetric encryption. It is mathematically associated with a private key, but is widely distributed. Public keys are frequently certified by a Certification Authority, so that users of this key can verify its authenticity [123].

Public-Key Certificate: Contains the name of a user, the public key component of the user, and the name of the issuer who vouches that the public key component is bound to the named user [7].

Public-Key Infrastructure: See PKI.

Purchase Mobile Agent: A mobile agent which like a window-shopping mobile agent collects price quotes for some item/s, but with the additional capability to make a purchase of the best priced item/s (at the end of window-shopping) or immediately a “bargain” price has been offered.

RA: “Registration Authority”, an entity trusted to register other entities and assign them a relative distinguished value such as a distinguished name or, a hash of a certificate. A registration scheme for each registration domain ensures that each registered value is unambiguous within that domain [211].

RBAC: “Role-Based Access Control”, a system of controlling which users have access to resources based on the role of the user. Access rights are grouped by role name, and access to resources is restricted to users who have been autho-
rised to assume the associated role. For example, if a RBAC system were used in a hospital, each person that is allowed access to the hospital’s network has a predefined role (doctor, nurse, lab technician, administrator, etc.). If someone is defined as having the role of doctor, than that user can access only resources on the network that the role of doctor has been allowed access to. Each user is assigned one or more roles, and each role is assigned one or more privileges to users in that role [107].

**RECDAM**: “Real-time Execution Checking and Deterrent Against Misbehaviour”, a novel *execution integrity* scheme designed for the MASHIn; RECDAM facilitates *real-time in-mission checking* and is a real deterrent against misbehaviour by either major stakeholder as their inputs into an agent’s execution session are non-repudiable (and administered by authoritative MASHIn TTPs).

**REV (Paradigm)**: “Remote Evaluation” Paradigm, a distributed processing model whereby a client user sends their own procedure code to a server, and requests the server to execute it and return the results. It may be argued that REV is nothing more than a special case of the client-server paradigm in which the server exports an *execute_code* service that takes a code fragment as parameter [214].

**Real Callback Class/es**: These are callback classes for mobile agents, only accessible to trusted entities (e.g. MASHs and checking hosts) in the agent’s itinerary. Sensitive code and data, not privy to untrusted agent platforms, can be encapsulated in these ‘real’ callback classes. They are part of the MASHIn’s *SCC4MAP* scheme for mobile agent code and data privacy.

**Real-time In-mission Checking**: In terms of mobile agent execution integrity checking, an execution check is performed immediately (as soon as viable) after executing at each agent platform in the agent’s itinerary. This mode of execution integrity checking has benefits for both major stakeholders including localisation of attacks, and is an integral part of the MASHIn’s *RECDAM* scheme.

**Reference States**: A reference state consists of the variable parts (i.e. the state) of a mobile agent executed by an agent platform showing reference behaviour. A
reference agent platform is one that acts as expected (i.e. is non-attacking). Thus an attack against an agent’s execution integrity can be detected by a difference in behaviour between the attacking agent platform and a reference agent platform. The reference states scheme for agent execution integrity was proposed by Hohl [86, 85], and improved upon in the MASHIn’s RECDAM execution integrity scheme.

**Registration Authority**: See RA.

**Remote Evaluation Paradigm**: See REV (Paradigm).

**Replay Execution Attack**: An attack whereby an agent platform re-executes an agent that it is holding captive to seek more information on its motives or perhaps to gain sensitive data from the agent.

**Reputation Model**: A model in a system or infrastructure whereby reputation data on some entities can be collected and subsequently analysed.

**Role-Based Access Control**: See RBAC.

**SCC4MAP**: “Secure Callback Classes for Mobile Agent Privacy”, a novel mobile agent data and code privacy scheme designed for the MASHIn; SCC4MAP is a highly-effective MA privacy scheme utilising ‘real’ and ‘dummy’ callback classes for disseminating sensitive versus non-sensitive mobile agent data and code.

**SSL**: “Secure Sockets Layer”, a protocol that encrypts a single TCP session. Using this asymmetric encryption, all data exchanged over a TCP socket can be cryptographically protected. SSL is the base of HTTPS - the secure World-Wide Web protocol [123].

**Second-hand Opinions**: These are the opinions gathered from secondary sources, for example a collective group of citizens or a friend of a friend. They are less reliable than unbiased trusted first-hand observations, and thus termed “opinions” rather than “observations”.
Secure Sockets Layer: See SSL.

Stationary Agent: An agent that once deployed, either locally or dispatched only once to a remote agent platform, does not migrate. It is stationary on that particular agent platform for its lifetime - thus, a stationary agent is not a mobile agent.

Symmetric Cryptography: Cryptography in which the same key (or a key easily derivable from the other) is used for both encryption and decryption [7].

TTL: “Time-To-Live”, the time before some session state/data should be invalidated.

TTP: “Trusted Third Party”, in general, an independent, unbiased third party that contributes to the ultimate security and trustworthiness of computer-based information transfers [211].

Time-To-Live: See TTL.

Trusted Third Party: See TTP.

VPN: “Virtual Private Network”, a network that is constructed by using public wires to connect nodes. For example, there are a number of systems that enable you to create networks using the Internet as the medium for transporting data. These systems use encryption and other security mechanisms to ensure that only authorised users can access the network and that the data cannot be intercepted (and easily reconstructed to legible a form) [107].

WAN: “Wide Area Network”, a physical or logical network that provides data communications to a larger number of independent users than are usually served by a local area network (LAN) and is usually spread over a larger geographic area than that of a LAN [7].

Wide Area Network: See WAN.
Window-Shopping Mobile Agent: A mobile agent whose mission is simply to collect price quotes for some item/s from vendor agent platforms.
Appendix B

SCC4MAP Proof-of-Concept Design

As discussed in Section 7.3.3, the SCC4MAP proof-of-concept prototype application - a purchase mobile agent - consisted of approximately 1500 lines of new code, encapsulated within the following new Java classes:

- **AgentServer**: The agent platform abstraction which listens for incoming agents; instances of *AgentServerThread* are created as needed to handle individual new (incoming agent) connections to the agent server.

- **AgentServerThread** extends *java.lang.Thread**: Handles one incoming agent connection, receives the agent jar from the previous networked agent platform server, writes the jar to local disk, and instantiates a *ReadyAgentForLoading* object to prepare the agent for interpretation on the agent platform.

- **ServerProps**: Defines the static properties of an agent platform server including whether it is modelling a MASHIn TTP (i.e. MASH or checking host) or untrusted agent platform, its network port for receiving agents, directories for storing ‘real’ and/or ‘dummy’ callback classes (as appropriate to the agent platform type), and directories for storing temporary files.

- **ReadyAgentForLoading**: Extracts the agent files from the encapsulating jar file, extracts the jar’s files to its own temporary working directory,
puts necessary files in a bootloader directory (consulted by a customised class loader), invokes the agent main/driver class (i.e. starts the agent thread of execution by instantiating a SimpleClassLoader object), prepares the dummy callback class for shipping (if the agent is a MASHIn TTP - i.e. MASH or checking host), serialises the agent ready for transport to the next agent platform, builds a new jar with all required files for transport to the next agent platform, and ships the agent to the next agent platform in its itinerary (by instantiating a ClientTransferJar object). This class is important for placing the correct version (i.e. either ‘real’ or ‘dummy’ callback class/es) in the local bootloader directory.

- SimpleClassLoader extends java.lang.ClassLoader: Like the standard Java classloader, determines (but not in the identical order) whether to grab a local cached version of a class file (if available), a system class file, or perform some other operation of loading (in our case loading the class from a local bootloader directory). In terms of loading callback classes, there is no intelligence specifically encapsulated in this class on whether the agent platform is loading a ‘real’ or ‘dummy’ callback class, as the creator ReadyAgentForLoading object has already deposited the correct callback class file type in the applicable local bootloader directory.

- PurchaseAgent implements java.lang.Runnable, java.io.Serializable: The prototype application-specific main/driver class first thread of initiation when the agent is started on an agent platform server. At some time, invokes (transparently to itself) either a ‘real’ or ‘dummy’ callback class - in this application these callback classes are stored in files named “CallbackClass-REAL” and “CallbackClass-DUMMY” respectively.

- CallbackClass-[REAL/DUMMY]: The two versions of callback classes; only MASHs and checking hosts are privy to the both the ‘real’ and ‘dummy’ callback classes - untrusted agent platforms only ever see the ‘dummy’ callback classes. In this application example, the ‘real’ callback classes have populated afterEveryHost() and afterMission() methods, whereas the ‘dummy’ callback classes have empty method implementations. See Section 7.3.2 and Section 7.3.3 for detailed information on the purpose of these methods and the prototype application-specific logic encapsulated in these methods respectively.
- **ClientTransferJar**: A class allowing reliable socket transport of an agent (encapsulated in a jar file) from one agent platform to another agent platform on its specific listening port. This class is also capable of updating the specific main/driver agent class (the suffix to this main/driver class changes in this prototype to differentiate versions of the agent as it does work on different agent platforms in its mission).

There were some simplifications taken in building the SCC4MAP proof-of-concept prototype mobile agent system and application (i.) there was no forwarding of ‘real’ callback classes - i.e. we simulated the inclusion of MASHs and checking hosts by returning the agent to the same one agent platform in-between visits to untrusted agent platforms, and (ii.) agent properties such as its itinerary were inflexibly prescribed - e.g. for simplicity, an agent’s next hop - i.e. next agent platform destination - was hard-coded in the server’s properties file.

Nevertheless, both of these simplifications can be overcome with trivial additional code functionality, and neither simplification effects the success of the proof-of-concept for demonstrating ‘real’ and ‘dummy’ callback class loading capability; the proof-of-concept positively revealed that the SCC4MAP strategy is a feasible method for effectively providing mobile agent data and code privacy.
Bibliography


[19] Ingrid Biehl, Bernd Meyer, and Susanne Wetzel. Ensuring the Integrity of Agent-Based Computations by Short Proofs. In Kurt Rothermel and


