Preemptive mobile code protection using spy agents

Georgios Kalogridis

Information Security Group
Department of Mathematics
Royal Holloway, University of London

Thesis submitted to The University of London

for the degree of Doctor of Philosophy

2011.

Declaration of Authorship

These doctoral studies were conducted under the supervision of Prof. Chris J. Mitchell.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with my supervisor, whilst enrolled in the Information Security Group of Royal Holloway, University of London as a candidate for the degree of Doctor of Philosophy. Where I have consulted the work of others, this is always clearly stated. This work has not been submitted for any other degree or award in any other university or educational establishment.

Signed: _____

Date: _____

Abstract

This thesis introduces 'spy agents' as a new security paradigm for evaluating trust in remote hosts in mobile code scenarios. In this security paradigm, a spy agent, i.e. a mobile agent which circulates amongst a number of remote hosts, can employ a variety of techniques in order to both appear 'normal' and suggest to a malicious host that it can 'misuse' the agent's data or code without being held accountable.

A framework for the operation and deployment of such spy agents is described. Subsequently, a number of aspects of the operation of such agents within this framework are analysed in greater detail. The set of spy agent routes needs to be constructed in a manner that enables hosts to be identified from a set of detectable agent-specific outcomes. The construction of route sets that both reduce the probability of spy agent detection and support identification of the origin of a malicious act is analysed in the context of combinatorial group testing theory. Solutions to the route set design problem are proposed.

A number of spy agent application scenarios are introduced and analysed, including: a) the implementation of a mobile code email *honeypot* system for identifying email privacy infringers, b) the design of sets of agent routes that enable malicious host detection even when hosts collude, and c) the evaluation of the credibility of host classification results in the presence of inconsistent host behaviour. Spy agents can be used in a wide range of applications, and it appears that each application creates challenging new research problems, notably in the design of appropriate agent route sets.

Acknowledgments

I would like to express my gratitude to my supervisor Professor Chris J. Mitchell, at Royal Holloway, University of London, to whom I am indebted. His precise and generous technical feedback, extensive editorial comments, and overall guidance have been invaluable to both me and this work.

I would also like to thank my advisor Dr. Allan Tomlinson at Royal Holloway, University of London, for his support.

This part time PhD study has been supported and funded by the Telecommunications Research Laboratory (TRL) of Toshiba Research Europe Limited, which has given me the opportunity to be part of a team leading in research and innovation. I would like to express my special thanks to Professor Joe McGeehan, Dr. Mutsumu Serizawa, and other Directors of Toshiba TRL for all the support and encouragement, without which this work would not have been possible.

Further, I thank my industrial supervisor Dr. Tim Farnham for this insightful technical comments, Dr. Mahesh Sooriyabandara for his support and motivation, and numerous other colleagues at TRL for the discussions that helped this work progress.

Last, but not least, I would like to dedicate this work to my family and in particular my parents, as well as my partner Jennifer. Their support and understanding will always be inspirational.

Contents

List of Figures

List of Tables

List of Acronyms

ACL	Agent Communication Language, 44
AEE	Agent Execution Environment, 42, 64
AES	Advanced Encryption Standard, 35
AI	Artificial Intelligence, 40, 41
BIBD	Balanced Incomplete Block Design, 79
CFF	Cover-free Families, 79
DNA	Deoxyribonucleic Acid, 77, 78, 83, 155, 156, 161, 184, 262, 267
DoS	Denial of Service, 35, 101, 245, 252, 253
DSA	Digital Signature Algorithm, 37
FIPA	Foundation for Intelligent Physical Agents, 44, 89
GT	Group Testing, 21, 74, 75, 78, 83, 129, 131, 132, 144, 150, 186–189, 197, 201, 205
GTC	Group Testing for Complexes, 78, 79, 154, 155, 157, 161, 166, 171, 173, 183, 197, 256, 266
ID	Identification Data, 57, 91, 102, 110, 250
ISP	Internet Service Provider, 70, 111
KDP	Key Distribution Pattern, 79, 161, 166, 180
MAC	Message Authentication Code, 37, 58
MASIF	Mobile Agent System Interoperability Facility, 43, 44

NGT Non-adaptive Group Testing, 76–78, 129, 133, 136, 186, 187, 197, 200, 201, 228, 234, 239, 251, 256, 266

OECD Organisation for Economic Co-operation and Development, 67, 69

PETs Privacy Enhancing Technologies, 25, 70
PII Personally Identifiable Information, 70, 133, 247, 251, 256
PKI Public Key Infrastructure, 39
P3P Platform for Privacy Preferences, 69
PRAC Partial Result Authentication Code, 57, 60

QoS Quality of Service, 71

RSA Rivest-Shamir-Adleman, 36, 37

SAEH Spy Agent Email Honeypot, 247–251, 255
SASH Spy Agent Shopping Honeypot, 252–256
SGT Sequential Group Testing, 75, 76, 185–190, 192, 197, 200, 228, 239, 251, 256, 266
SQL Structured Query Language, 44

TSP Trust Service Provider, 65
TTP Trusted Third Party, 39, 60

VM Virtual Machine, 42, 61

"Every new beginning comes from some other beginning's end."
Seneca (4 BC – 65 AD)

1 Introduction

Contents

1.1 Synopsis

This chapter provides an overview of the thesis. It discusses the rationale for the concept of *spy agents* (§1.2), introduces the motivation for the work

described and the research challenges it faced (§1.3), and outlines the structure of the thesis along with the main contributions (§1.4).

1.2 Setting the scene

Preemptive protection of a computer network involves monitoring network and host activity in order to detect policy violations and anomalous behaviour, identify the origin of such activity, and make security evaluations. These evaluations can be used to formulate and report on an appropriate reaction, in an attempt to address the detected security threats. For example, preemptive host and network protection against *mobile code* aims to identify (or evaluate the risk of) *software viruses*, *trojan horses*, *malware*, etc. Approaches to support such measures include *intrusion detection systems*, *honeypots* and *software decoys*. Preemptive protection does not necessarily involve security controls that thwart security attacks—it also includes measures for evaluating, analysing and preventing their occurrence.

This thesis focuses on the protection of mobile code, inherently a harder challenge than the protection of the mobile code hosts. We also assume that malicious hosts are 'intelligent' in the sense that they mount attacks selectively and that they are most likely to mistreat mobile code when it appears to them that they will not be identified. Thus, the challenge is to find techniques that 'outwit' such malicious hosts. Spy agents, proposed in this thesis, address this challenge through the use of Group Testing (GT) techniques.

The spy agent paradigm (see Chapter 3) involves the provision of a new kind of security service designed to indirectly improve the overall security level. It achieves this by using evaluation mechanisms that preemptively assess the trustworthiness of remote hosts, before they are sent vulnerable mobile code.

We introduce the notion of spy agents as 'legitimate' *mobile agents* which are able to interact with remote, potentially hostile, mobile agent platforms in a manner that helps to enable trust assessment.

Spy agents provide an auxiliary layer of defence. Such a layer is of value since current security solutions do not offer complete protection; this issue is discussed further immediately below, while a literature review is given in Chapter 2. In general, spy agents may co-exist with, or even require the functionality of, other security solutions; this may serve to increase the effectiveness of each layer of defence.

☞ Throughout this thesis we (by convention) use the terms 'mobile agents' and 'mobile code' interchangeably.

1.2.1 Mobile agent security issues

Mobile agents are the basis of a distributed programming infrastructure with inherently beneficial characteristics such as autonomy, flexibility and intelligence [69]. One widely cited example of an application of agent technology is a price comparison agent which 'visits' a number of on-line retailer sites or nodes and requests a price for a particular item. This agent could retrieve and process information, including, for example, prices, from a number of different retailers.

The two main actors in a mobile agent system are the following:

- Agent: an instance of mobile code;
- Host: a platform capable of executing an agent.

Mobile agent security has been the subject of a considerable volume of

recent research [89] (see §2.4). As has been widely discussed (see, for example, [18, 89]) there are three parallel sets of security issues associated with mobile agents, namely protecting hosts (and other agents) against malicious agents, protecting agents against malicious hosts, and protecting hosts (and other agents) against third parties.

In a typical mobile agent system, legitimate mobile agents will interact with hosts in a defined way, and hosts will be built to deal with expected agent behaviour. Viruses and other 'illegitimate' agents may attempt to access the host in unauthorised ways rather than remain in the execution environment reserved for agents (e.g. a *sandbox*). Such malicious agents might steal sensitive information from the host, such as personal financial details, cause the host to act in an unintended way, for example causing it to send *spam* emails, or simply corrupt the host so that it no longer functions properly.

The parallel security problem, and the main focus of this thesis, arises from the fact that agents are at the mercy of the host which executes them. Ultimately a host can choose to carry out the functions requested by the agent as expected, and/or it can manipulate the agent. Such manipulation might include reading data contained within the agent which is intended to remain private, e.g. the source address or the identity of the agent originator. This information could then be misused for a variety of purposes, including forwarding spam to the originator's email address. Other examples of inappropriate behaviour might include reading quotes from competitor on-line retailers and providing a more attractive quote, or changing competitor quotes to make them less attractive.

Autonomous mobile agents, apart from obtaining price quotes or retrieving

other information for further analysis, might also be able to complete a transaction remotely in accordance with the client's instructions. For example, to purchase an air ticket, an agent might be instructed to visit several on-line stores before automatically making the best value purchase. The client might only wish to pass a third party certain personal information embedded in the agent if there is a considerable discount on the final price. Giving an agent power to make commitments on a client's behalf clearly permits a range of possible abuses by a host, as further discussed in §2.4.2.

1.2.2 Host protection

In this thesis we do not address techniques for host protection; nevertheless, host protection techniques are relevant here since they can provide input to spy agent security evaluation scenarios. Hosts can use a variety of techniques to protect themselves against malicious mobile code. For example, 'safe' programming languages can restrict mobile code permissions; a sandbox (such as a Java security sandbox [136, 187]) can enforce a security policy for code execution; and *proof carrying code* can provide a formal (authenticated) proof that received code will execute as expected [125].

Host protection mechanisms are reviewed in §2.4.4.

1.2.3 Mobile code protection

As mentioned in §1.2.1, in this thesis we consider the threat posed to mobile code by malicious hosts. This is an important research area, and many schemes to provide protection for code have been proposed (including *tamper-proof hardware* [199], *tamper-proof execution environments* [189], *code obfuscation* [10], *encrypted functions* [145,152], and strategic division of functionality across multiple agents [127]). Nevertheless, none of the existing solutions is

able to address the problem in both a practical and robust manner.

Previously proposed mobile code protection mechanisms are reviewed in §2.4.3.

1.2.4 Mobile code data privacy

One particularly important threat posed by a remote host to a mobile agent is that it might breach its data privacy policy. A personal email address is an example of private data whose use might be protected by a privacy policy. Infringements of email address storage and usage policies could result in a range of undesirable effects, including the receipt of unsolicited messages, i.e. spam, identity theft, and *phishing*.

The misuse of sensitive private data contained in mobile code is a passive attack, which leaves no signs of violation in the mobile code itself and may only be detected indirectly (e.g. by checking an email address for the receipt of spam). Current detection systems, such as Privacy Enhancing Technologies (PETs), monitoring agents and honeypots (discussed in §2.5), cannot be used to identify unauthorised use of remote code/data, as in the latter case the personal information is willingly exposed to multiple hosts.

1.3 Motivation and challenges

In order to help address the mobile code security issues discussed above, this thesis introduces spy agents as a new preemptive mobile code protection paradigm. The main motivation for proposing these agents is that current security controls are limited in scope and application. We seek to design spy agent systems that provide pragmatic remote security evaluations—one major

challenge is that 'sophisticated' attackers may escape detection by misbehaving selectively, i.e. only misbehaving when they can avoid detection.

Instead of trying to mitigate the mobile code threats discussed above directly, the challenge addressed is to develop a robust trust evaluation mechanism that can indirectly help to provide security for mobile agents. The mechanism we develop employs special mobile agents to retrieve and process security related information from target hosts. This concept was discussed previously by Borselius et al. [19], who suggest that a security assessment can be made when an agent migrates from a trusted platform to a target platform, where it obtains certain information and then returns to the trusted host for further analysis.

A wide variety of preemptive mobile code protection paradigms have been proposed; the idea of a distributed security system made up of security agents patrolling target hosts to monitor their behaviour is not novel. However, in previous approaches [26,28,40,67] such security mechanisms or agents operate in trusted environments and cannot be applied to evaluate remote, potentially hostile, domains, where some hosts may exhibit complex malicious behaviour. For example, in the scheme of Vogler, Spriestersbach and Moschgath [188], part of an agent needs to run in a tamper-resistant environment, which is trusted to perform sensitive operations.

The requirement for a trusted environment is relaxed in [19], where it is suggested that a security assessment can be made as a result of agent *migration* to unknown hosts; however, this approach still has limitations. The main problem is that it is assumed that the target hosts will adhere to their policies and will provide the agents with all the security assessment information they request. Hence a potentially corrupted remote host could provide security

agents with apparently proper information. The host could later behave inappropriately when it has the opportunity to do so without being detected; also, in order to escape detection, it might only selectively misbehave if it knows it is being monitored.

In contrast with this previous work, spy agents are designed to be able to obtain information that reflects the genuine character of a remote host. The main functional constraint is that information should be acquired without violating the hosts' own security and privacy controls, regardless of whether or not a host is malicious. Given this, the main challenge addressed by this thesis can be formulated as follows. Spy agents should assess the genuine character of remote hosts by providing the hosts with no indication of their purpose and by interacting with visited hosts in the expected way.

Spy agents are analogous to software decoys [118] and honeypots [139] in that they hide any indication of their purpose from a target. A spy agent differs, however, in that it is mobile code which is fully under the control of a remote host. It is interesting to observe that honeypots, software decoys and other intrusion detection technologies can be used by malicious hosts against spy agent deployments. Spy agents must address this challenge, and must still be able to identify sophisticated malicious hosts that may adapt their behaviour to avoid detection.

In some ways, spy agents are analogous to *cryptographic tracing* techniques, which can be used to detect unauthorised modifications to an agent's code in a multi-host route [185]. A trace is a log of the operations performed by an agent during its lifetime. In a tracing scenario, platforms are required to create, maintain and exchange authenticated logs for all incoming agents. After an agent has terminated, its owner can acquire all the logs and compare

them against the logs produced locally to detect discrepancies. However, this approach has the following limitations.

- If the language adopted for agents allows for the processing of very complex data structures, modifications to the agent's internal state could be difficult to represent in a log entry and require a lot of space, rendering the mechanism infeasible in practice [185].
- Privacy issues are likely to arise for secure log management.
- Only attacks involving illegal modification of code, state and execution flow of a mobile agent can be detected. That is, indirect attacks, such as misuse of a personal email address, cannot be detected.

The research challenge involved in designing the spying approach is to allow unaware malicious hosts to abuse 'normal' (spy) agents (e.g. without requesting proofs), and then to be able to infer the level of trustworthiness of the visited hosts from the limited information (signs of abuse) available after agent termination.

1.4 Thesis structure and contributions

The material in the remainder of this thesis is organised as follows.

Chapter 2 covers the background material necessary for the thesis.

Chapter 3 introduces the core contribution of this thesis: it presents the spy agent concept and describes the spy agent system and framework.

Chapter 4 expands on the previous chapter by defining and analysing fundamental spy agent system requirements and system implementation assumptions.

Chapter 5 considers a number of fundamental spy agent routing protocol architectures, and defines the main system parameters.

Chapter 6 focuses on the problem of designing spy agent routes and evaluating the results from the deployment of a set of spy agents; a mathematical formulation of the design and analysis problem is given using group testing theory.

Chapter 7 extends the analysis given in Chapter 6 to the case where complex malicious behaviour and collusion by hosts may occur; a novel group testing algorithm is described which enables hosts exhibiting complex malicious behaviour to be detected (under certain reasonable assumptions about their behaviour).

Chapter 8 discusses an alternative spy agent scenario, and an optimal multi-stage routing algorithm for host testing for this scenario is introduced.

Chapter 9 proposes and studies a probabilistic model for evaluating the credibility of spy agent host classification results on the assumption of inconsistent host behaviour; further, it introduces a methodology for analysing the error-resilience properties of spy agent route designs.

Chapter 10 considers two important practical applications of the spy agent paradigm, namely identifying data privacy infringements and identifying code *black box* attacks.

Chapter 11 concludes this thesis and identifies possible directions for future research.

1.5 Publications

Some of the results contained in this thesis have been published in papers and patents, as listed below.

1.5.1 Conference papers

- G. Kalogridis. Protecting Mobile Code Privacy With Resilient Spy Agent Group Testing. In C. A. Ardagna, S. De Capitani di Vimercati, C. D. Jensen, and R. Küsters, editors, *Proceedings of the Fifth International Workshop on Security and Trust Management (STM '09)*, Electronic Notes in Theoretical Computer Science (ENTCS), pages 58–71, Saint Malo, France, September 2009. Elsevier.
- G. Kalogridis and C. J. Mitchell. Using nonadaptive group testing to construct spy agent routes. In S. Jakoubi, S. Tjoa, and E. R. Weippl, editors, *Proceedings of the Third International Conference on Availability, Reliability and Security (ARES '08)*, pages 1013–1019, Barcelona, Spain, March 2008. IEEE Computer Society.
- G. Kalogridis, C. J. Mitchell, and G. Clemo. Spy agents: Evaluating trust in remote environments. In Hamid R. Arabnia, editor, *Proceedings of the 2005 International Conference on Security and Management (SAM '05)*, pages 405–411, Las Vegas, Nevada, USA, June 2005. CSREA Press.

1.5.2 Patents

- G. Kalogridis. Method for identification of insecure network nodes. United Kingdom Patent Office (UKPO), GB2452555, March 2009. Assignee: Toshiba Research Europe Limited.

- G. Kalogridis. Network node security analysis using mobile agents. United Kingdom Patent Office (UKPO), GB2428315, January 2007. Assignee: Toshiba Research Europe Limited.
- G. Kalogridis. Network node security analysis method. United Kingdom Patent Office (UKPO), GB2415580, December 2005. Assignee: Toshiba Research Europe Limited.

"Every skilled person is to be believed with reference to his own art."

Legal maxim

2 Background

Contents

2.1 Synopsis

This chapter gives the technical background for the thesis. Each section of this chapter, as listed below, can be read independently.

- §2.2 provides key definitions and notation for the cryptographic primitives used throughout this thesis.
- §2.3 provides an introduction to mobile agents and mobile code; this gives the necessary background for the spy agent system introduced in Chapter 3.
- §2.4 reviews the state of the art in mobile agent security, and provides a framework for the main contributions of the thesis.
- §2.5 discusses other related work, including data privacy, preemptive trust assessment, and deception-based security mechanisms.
- §2.6 introduces the theory of group testing, and also provides related background material on combinatorial designs. This is used in Chapters 6, 7 and 8 to help in the design of sets of spy agent routes and agent evaluation mechanisms.

2.2 Cryptographic primitives

2.2.1 Preliminaries

In this section we introduce some fundamental *information security* terms used throughout the thesis. Dent and Mitchell [41] provide the following useful notions. "A *security threat* is something that poses a danger to the security of a system. A *security service* is selected to meet an identified *threat*, and a *security mechanism* is the means by which a service is provided or implemented."

Definitions for standard security services and security controls for data communication systems can be found in [87]; standard information security terminology can be found in [61,66]; and specifications of security controls with industrial strength can be obtained from standards published by ISO/IEC JTC 1/SC 27[1]. We provide key definitions below, adapted from [41] and and ISO 7498–2 [83].

Definition 2.1. Entity authentication: *this service is the corroboration that the entity at the other end of a communications channel is the one claimed.*

☞ Entity authentication addresses spoofing threats.

Definition 2.2. Data origin authentication: *this service is the corroboration that the source of data received is as claimed.*

☞ Data origin authentication addresses tampering threats.

Definition 2.3. Data confidentiality: *this service is concerned with preventing the disclosure of data to unauthorised entities.*

☞ Data confidentiality addresses information disclosure threats.

[1]Subcommittee 27 (IT security techniques) of Joint Technical Committee 1 (Information technology) of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).

Definition 2.4. Data integrity: *this service is concerned with preventing unauthorised alteration or destruction of data by an unauthorised entity (in other words unauthorised tampering with data).*

☞ Data integrity addresses tampering threats.

Definition 2.5. Accountability *or* **non-repudiation***: this service is concerned with preventing denial by an entity that it has taken a particular action, e.g. sending or receiving a message.*

☞ Accountability addresses repudiation threats.

Definition 2.6. Access control: *this service is concerned with preventing unauthorised use of a resource.*

☞ Access control addresses elevation of privilege threats.

Definition 2.7. Availability: *this service ensures that computer system assets are available to authorised parties when needed.*

☞ Availability addresses Denial of Service (DoS) threats.

2.2.2 Cryptographic functions

Cryptographic functions are security mechanisms that can be used to provide security services. In this section we introduce the main cryptographic functions we use in this thesis, using definitions taken from Menezes, van Oorschot and Vanstone [117].

2.2.2.1 Symmetric and asymmetric encryption

Encryption helps to provide confidentiality services. An encryption scheme consists of two sets of encryption and decryption transformations $\{E_e : e \in \mathcal{K}\}$ and $\{D_d : d \in \mathcal{K}\}$, respectively, where $\mathcal{K}$ is the keyspace.

An encryption scheme is said to be *symmetric-key* if, for each associated encryption/decryption key pair (e, d), it is computationally 'easy' to determine d knowing only e, and to determine e from d. In most practical symmetric-key encryption schemes $e = d$; examples of such schemes include *block ciphers*, such as the Advanced Encryption Standard (AES) [123], and *stream ciphers*, such as the RC4 algorithm [146].

In contrast with symmetric encryption, an encryption scheme is said to be *asymmetric* if, for each associated encryption/decryption key pair (e, d), it is computationally infeasible to compute d from e. Typically, encryption will use a *public key* (i.e. a key e that is widely known), and decryption will use a *private key* (i.e. a key d that is kept secret by its owner); examples of such schemes include the Rivest-Shamir-Adleman (RSA) scheme [147] and the ElGamal scheme [53].

2.2.2.2 Hash functions

A (cryptographic) hash function helps to provide integrity and authentication services. It is a computationally efficient function that maps binary strings of arbitrary length to binary strings of some fixed length, called *message digests* or *hash values*, where this function has the following properties:

1. *Pre-image resistance*: given any output, it is computationally infeasible to find an input which maps to that output.

2. *Second pre-image resistance*: given any input, it is computationally infeasible to find a second input which yields the same output.

3. *Collision resistance*: it is computationally infeasible to find two inputs which map to the same output.

An example of a hash function is SHA-256 [133] (32-byte output).

2.2.2.3 Message authentication codes

A Message Authentication Code (MAC) function can be used to provide data origin authentication and integrity services. A MAC function takes as input a message and a secret key, and outputs a fixed length binary, called a MAC value (or simply a MAC). The secret key can be used to recompute the MAC and verify that the original message has not been tampered with.

Examples of MAC functions include CBC-MACs [85] and HMAC [86].

2.2.2.4 Digital signatures

A *digital signature* can be used to help to provide authentication, non-repudiation, and integrity services. Its purpose is to provide a means for an entity to bind its identity to a message. A digital signature scheme consists of a signing function S_A and a verification function V_A for an entity A. The signing function S_A is a transformation from a message set $\mathcal{M}$ to a signature set $\mathcal{S}$. This transformation uses a signature key which should be kept secret by A and should only be used for the purpose of creating signatures. The verification function V_A is a transformation from the set $\mathcal{M} \times \mathcal{S}$ to the set $\{true, false\}$. This transformation uses a verification key which is typically publicly known, and which can be used to verify the signatures created by A.

Examples of digital signature schemes include the RSA signature scheme [117, p. 433], the Fiat-Shamir signature scheme [117, p. 447], and the Digital Signature Algorithm (DSA) [117, p. 451].

2.2.3 Key establishment, management and certification

Key establishment is the process of setting up a secret key shared by two (or more) parties for subsequent cryptographic use. *Key management* refers to the set of processes supporting key establishment and the maintenance of ongoing keying relationships between parties, including key substitution, key recovery, and key revocation.

Key establishment and key management schemes typically involve the use of cryptographic protocols. A *cryptographic protocol* is a distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more entities to achieve a specific security objective. As a more general term, a *security mechanism* is a technique encompassing protocols, (single-entity) algorithms and other (non-cryptographic) controls to achieve specific security objectives.

A *key transport protocol* is a special class of cryptographic protocol which supports the secure transfer of a secret key. An example of such a protocol using digital signatures and public-key encryption and which enables entity A to send a secret key k to entity B, is as follows, [117, p. 510].

$$A \to B: \ \ {t_A}^*, P_B\left(i_A, k\right), S_A\left(i_B, {t_A}^*, P_B\left(i_A, k\right)\right) \ ,$$

where i_A and i_B are identifiers for A and B, t_A is a timestamp created by A, the asterisk denotes that t_A is optional, P_B is an asymmetric encryption function that uses the public key of B, S_A is a digital signature function that uses the private key of A, and the comma denotes message concatenation.

The main purpose of the above protocol is to provide integrity, confidentiality and authentication services for the communicated key. The (optional) timestamp t_A provides guarantees regarding message 'freshness'. Further, the

identifier i_A within the scope of P_B prevents *signature stripping*. That is, it prevents a third party C from sending B the message

$$t_A{}^*, P_B(k), S_C(B, t_A{}^*, P_B(k)),$$

thus defeating authentication by means of impersonation. We note that the above protocol is a one-pass protocol that does not provide authentication of B to A. For mutual authentication it is necessary to employ more advanced protocols, such as *challenge-response* protocols.

The above key transport protocol depends on the existence of asymmetric key pairs for A and B. This gives rise to one of the fundamental key establishment problems, namely the distribution of keys in such a way that the recipients can verify that they are genuine. One well-known key establishment scheme is the Diffie-Helman (DH) protocol [20, Chapter 5]. However, the DH protocol does not offer authentication, and is susceptible to man-in-the-middle attacks. Other examples of key establishment protocols can be found in [117, Chapter 12].

When using public key cryptography (including asymmetric encryption and digital signatures) the fundamental key management problem is the reliable distribution of user public keys. One mechanism for achieving this is known as a Public Key Infrastructure (PKI) [2,170]. A PKI typically involves the use of a special purpose Trusted Third Party (TTP) called a Certification Authority (CA). A CA is responsible for issuing (digitally signing) public key certificates, which bind a public key to the name of its owner (together with other relevant information). In order to validate a certificate a user must have access to a trusted copy of the relevant CA's public key, in order to verify the signature. A CA may revoke a certificate by including it in an Certificate Revocation List (CRL), or by using the Online Certificate Status Protocol (OCSP) [2]. A

widely adopted standard for digital certificates is X.509 [88].

2.3 Mobile agents

2.3.1 Introduction to mobile agents

Mobile agents were briefly introduced in §1.2.1. They are mobile software components in network systems, and they can be used to participate in dynamic multi-party interactions. Mobile agent architectures are an alternative to 'standard' *client-server* software architectures, and they have been introduced with the goal of providing more efficient networking and enhanced application functionality.

Typical mobile agent applications include information filtering, electronic commerce, education and entertainment. Such applications, for example, require network functionality for advertising, finding, combining, using, presenting, managing and updating information.

The term *agent* appears to have been introduced in computer science in the context of Artificial Intelligence (AI) [74]. However, the term agent is today used in a wide range of contexts; typically, agents are "active, persistent (software) components that perceive, reason, act, and communicate" [80]. Some authors add further properties, such as *autonomy*, *rationality*, *reactivity*, or *proactivity*, whereas others limit agents to the role of representing a user or database. Agents can also be perceived in different ways. For example, some discuss agents as if they are *conscious* and *cognitive* entities that have *feelings*, *perceptions*, and *emotions* emulating humans, whereas others treat agents merely as *automata* that behave in expected ways as designed or programmed.

Adding the term *mobility* to some of the above features, Braun and Rossak

give the following two definitions [22].

AI view: *"Mobile software agents are computer programs that act as representatives in the global network of computer systems. The agent knows its owner, knows his or her preferences, and learns by communicating with its owner. The user can delegate tasks to the agent, which is able to search the network efficiently by moving to the service or information provider."*

Distributed systems view: *"Mobile agents refer to self-contained and identifiable computer programs, bundled with their code, data, and execution state, that can move within a heterogeneous network of computer systems. They can suspend their execution on an arbitrary point and transport themselves to another computer system. During this migration the agent is transmitted completely, that is, as a set of code, data, and execution state. At the destination computer system, an agent's execution is resumed at exactly the point where it was suspended before."*

There is an extensive literature on mobile agents. A useful collection of papers has been put together by Bradshaw [21]; an introduction to agent topics and applications can be found in [34]; and another useful collection of papers is given by Huhns and Singh [80].

Agents are typically implemented as components of a *multiagent system*. For example, resource agents might advertise services and user agents find and interact with (querying or informing) the resource agents. Multiple user agents might either collaborate in finding and combining information, or compete for resources. Similarly, service agents may either collaborate with or compete for (with) users, resources, and other service agents.

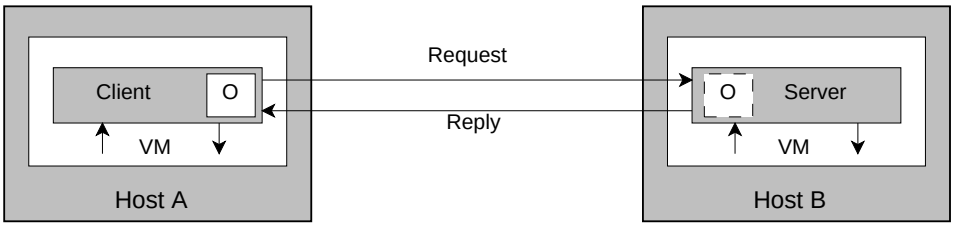


Figure 2.1: Client-server paradigm architecture.

Further information regarding multiagent and distributed AI systems can be found in [59] and [191].

2.3.2 Computing architectures

The traditional *client-server* paradigm is illustrated in Figure 2.1. The client code (executing within a Virtual Machine (VM) of Host A) makes a request to server code (executing within the VM of Host B). The server code may in turn access a resource before replying to the client code. Access to a resource is controlled by the VM, which could be an *Operating System*. Typical *client-server* programming language constructs include the *Remote Procedure Call* and the `Java` *Remote Method Invocation*.

In more complex paradigms, a client request may contain *dynamic code* to be executed at the server. In this case, the dynamic code would typically be encapsulated in the initial request message (Figure 2.1). A widely used example of the dynamic code paradigm is provided by the `PostScript`® language for printers.

A client could also request dynamic code from a server; in such a case the dynamic code would typically be encapsulated in the server's reply message (Figure 2.1). An example of this paradigm is provided by web clients fetching and executing *Java Applets*.

The effect on the client-server architecture of a shift to the mobile agent

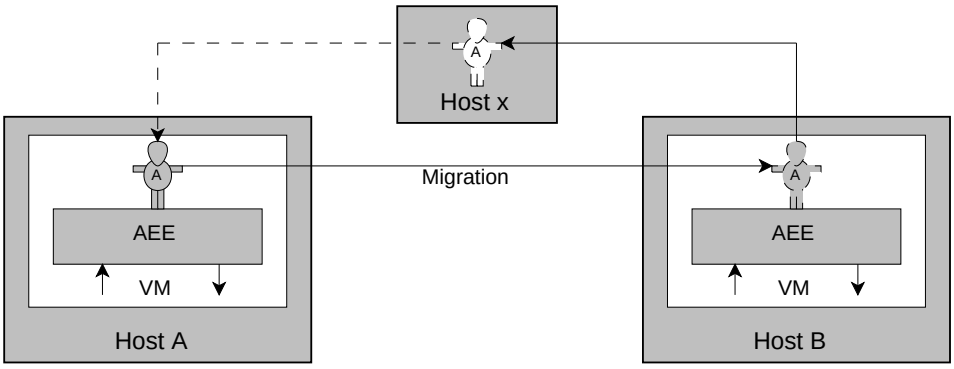


Figure 2.2: Mobile code paradigm architecture.

paradigm, as described in §2.3.1, is illustrated in Figure 2.2. The mobile agent is shown as an autonomous code component (A) which runs on an Agent Execution Environment (AEE) within the VM of a host. The mobile agent can then request to be migrated to another host, and so on.

2.3.3 Mobile agent implementations

For agents to operate, communicate, and migrate successfully it is essential that they have a common representation, i.e. that they 'speak the same language'. This is usually provided by a 'common ontology', which is a representation of knowledge in some domain of discourse that is available to all the agents [124]. Existing examples of ontologies and semantic mappings include the DARPA ontology sharing project [137], and Princeton's WordNet [119].

Complete mobile agent systems have been implemented and are available as mobile agent toolkits—some widely discussed examples are listed in Table 2.1. More comprehensive lists of mobile agent toolkits can be obtained from online sources. *IEEE distributed systems online*[2] lists available agent toolkits, while the AgentLink project[3] maintains a list of ongoing mobile agent projects.

[2] `http://computingnow.computer.org`
[3] `http://www.agentlink.org`

Table 2.1: Examples of mobile agent toolkits.

Toolkit	Organisation	URL
ADK	Tryllian	www.tryllian.com
Aglets	IBM (Open Source)	aglets.sourceforge.net
Ajanta	Univ. of Michigan	www.cs.umn.edu/Ajanta
D'Agents	Dartmouth College	agent.cs.dartmouth.edu
Grasshopper	IKV	www.grasshopper.de
Semoa	Fraunhofer Society	www.semoa.org
Bee-gent	Toshiba	www.toshiba.co.jp/rdc/beegent
Tracy	Univ. of Jena	www.mobile-agents.org

The Mobile Agent System Interoperability Facility (MASIF) is a standardisation body for mobile agent systems that has been promoted by a number of organisations including IBM, GeneralMagic, GMD Fokus, and OMG. MASIF defines interfaces for interoperable mobile agent toolkits; however, it relies on the use of CORBA[4] for agent communications.

The Foundation for Intelligent Physical Agents (FIPA)[5] is a parallel standardisation body for agent technology. FIPA attempts to address a broader range of agent interoperability issues than MASIF, including agent communication, message transport protocols, ontologies, and mobility (migration).

Interoperable mobile agents supplied by different implementors may communicate with each other using a common Agent Communication Language (ACL), such as one of the following.

- **KQML**: The Knowledge Query and Manipulation Language (KQML) assumes a layered architecture, using common communication protocols at the bottom, common application functionality at the top (e.g. Structured Query Language (SQL)), and, in the middle, agent specific primitives [60].

[4] http://www.omg.org/spec/CORBA/
[5] http://www.fipa.org

- **Arcol and FIPA**: Arcol was the basis for the first FIPA standard, discussed above.

2.3.4 Mobile agent applications

In general, applications built using mobile agents could also be implemented using traditional client-server techniques, as discussed in §2.3.2. However, the use of mobile agents is likely to offer advantages when it can improve the efficiency of network resource usage and reduce the complexity of developing applications.

Mobile agents are thought to be best-suited for open information-rich environments made up of a large number of distributed heterogeneous resources, in which applications require delegation of tasks, asynchronous processing and service adaptation. These characteristics make agents useful for efficient resource discovery, information retrieval, database queries, information filtering and data fusion.

As an example we consider the *Warren system*, which consists of intelligent agents that support financial portfolio management [176]. In particular, some agents are designed to combine market data, financial report data, technical models, analysts' reports, and breaking news with current prices from a 'stock ticker'; others integrate all the information; (ultimately) yet others present the processed information or create alerts. Agents may also take actions based on learned user preferences.

A review of mobile agent e-commerce applications can be found in [115]. As stated in a recent study [70], the success of mobile agents depends on their trustworthiness, which in turn depends on the existence of adequate security controls. This is the focus of the next section.

☞ In this thesis we do not adhere to any specific mobile agent definition, implementation, computing architecture, toolkit, or standard. The spy agents we introduce in the main part of this thesis can be implemented using a range of technologies; the exact agent functionality and implementation will depend on the context and scenario requirements. Also, we regard a spy agent system as a multiagent system within which spying information is acquired from untrusted agent hosts before it is combined for analysis in a trusted computing environment.

2.4 Mobile agent security

Mobile agent security was briefly introduced in §1.2.1. In this section we review known threats, security requirements, and security solutions, focusing primarily on mobile agent protection and secondarily on host protection.

2.4.1 Analysis principles

The process of evaluating the security of a system typically involves identification and analysis of applicable threats and vulnerabilities, so that appropriate security controls can be applied. The following approaches can be used:

- Top-down approach: threats and vulnerabilities are identified for specific scenarios.
- Bottom-up approach: 'common' security requirements (such as integrity, confidentiality, authentication, accountability, availability, and anonymity) are analysed for the various system components and functions.

2.4.2 Security threats and requirements

2.4.2.1 Threats

Threats to mobile agent security were briefly discussed in §1.2.1. An adversary might be able to compromise an agent using one or more of the following approaches.

- Observing the agent's code, data and flow control.
- Manipulating the agent's code, data and flow control.
- Executing the agent's code inappropriately, including re-execution.
- Returning false values to the agent's system call functions.
- Denying execution of the agent's code, either in part or whole.
- Masquerading as a different host.
- Eavesdropping on agent communications.
- Manipulating agent communications.

A similar and more elaborate classification of security threats for shopping agents (and corresponding security requirements and controls) is given by Schaefer [155]. Further discussions can be found in [17, 71, 75, 89]. Key classes of threats are discussed in greater detail below.

2.4.2.1.1 Attacking agent data

Yee [200] has studied the threats to mobile agent data arising in an e-ticketing scenario, in which agents collect offers from merchants. We follow Yee's classification.

An agent can carry either static (unmodifiable) or dynamic (modifiable) data, which may be either public or private. Public static data (e.g. user ticket

query information) require integrity protection, whereas private static data (e.g. parameters for price negotiations) also require confidentiality protection. Malicious hosts that access or modify such data could compromise market competition rules as well as customer security.

The use of encryption may not suffice to protect agent private data. For example, a malicious host might use a *cut-and-paste* attack in which the encrypted agent data is inserted into a new agent with code that instructs it to migrate to a host authorised to access the data; the agent then instructs the host to decrypt the encrypted data and return it (unencrypted) to the malicious host.

2.4.2.1.2 Attacking agent code

As agent is (by definition) under the control of the host that executes it; this makes it hard to maintain code confidentiality. A list of attacks of this type can be found in [200].

In a *black-box attack* an agent is executed by a malicious host multiple times, each time with different input parameters, with the goal of understanding its logic and state by observing its behaviour under a wide range of conditions. For example, in the e-ticketing scenario, a malicious host could gradually improve its offer until it is (marginally) better than previously received offers (and this is revealed by the agent behaviour). Alternatively, a malicious host might analyse the agent's code and attempt to understand its semantics.

A malicious host could also 'sabotage' the agent code by arbitrarily modifying it, or by inserting malware to cause it to behave in unintended ways.

Inserted malware could, for example, allow a malicious host to remotely control the agent after it migrates elsewhere. Alternatively, agent code could be temporarily changed; for example, execution could be manipulated to omit security checks or other conditional behaviour.

2.4.2.1.3 Truncation attacks

Truncation is an attack on agent data, code, or a combination of both, in which a malicious host removes information from the agent that was added by previously visited hosts (e.g. price offers). One example of such an attack would involve two malicious hosts colluding in order to remove all data from an agent that was inserted by hosts visited between visits to the two malicious hosts. Such an attack may be hard to detect if the agent itinerary is not fixed (e.g. for free roaming agents).

Specific types of truncation attacks include the following. In a *growing a fake stem attack* a malicious host selectively removes genuine offers and appends fake offers [103]; in a *revisiting attack* a malicious host modifies the agent's migration logic so that the agent will revisit it (which could, for example, allow this host to truncate all succeeding offers) [202]; in an *interleaving attack* a malicious host creates a new agent that appears to originate from a legitimate source by selectively combining information from one or more previous hosts [149], an attack similar to the cut-and-paste attack discussed above.

2.4.2.2 Security requirements

A summary of mobile agent security requirements is given below. This list is based on discussions in [17, 143].

- Agent authentication and authorisation: the origin and integrity of mobile agents should be verified, and agent access to host resources should be subject to an authorisation check.
- Situatedness: an agent should be aware of the environment it is executing in, and be able to apply the necessary security controls.
- Autonomy and migration: an agent should have control over its internal state and migration. Greater degrees of autonomy and more sophisticated migration capabilities require higher levels of security as a result of the increased risks arising from agent code manipulation.
- Communication: agent communication with its environment (i.e. other agents, hosts or humans) needs to be protected. Confidentiality, integrity, authenticity, non-repudiation and availability services must be provided for communicated data.
- Rationality, veracity, and benevolence: the agent should act in an expected way (and not act maliciously).
- Anonymity: while knowledge of the identity of an agent may be important for certain applications and services, it may not be needed by others.
- Trust: agents need to be capable of assessing the trustworthiness of received information [27] (e.g. by using a reputation system [156]).
- Delegation: it must be possible for an agent to be granted rights to carry out certain tasks on behalf of another entity. The security for such a delegation act could, for example, be supported by the use of public key and attribute certificates.

2.4.3 Security controls

We divide our discussion of security controls for mobile agents into three main classes, as follows [22].

Collateral techniques: These controls restrict the operations of agent systems for security purposes—they neither prevent nor detect attacks.

Prevention techniques: The purpose of these controls is to prevent security attacks from occurring.

Detection techniques: The purpose of such controls is to identify an attack and (or) the attacker after the attack has occurred.

2.4.3.1 Collateral techniques

2.4.3.1.1 Trusted host itinerary

When using a trusted host itinerary, hosts will not accept agents coming from, and will not despatch agents to, untrusted hosts (such functionality has been implemented in the `Aglets` agent system). In some implementations this limitation can be relaxed by allowing agents to visit untrusted hosts if they do not contain sensitive data or code; for example, an agent could be designed so that secure computations are only executed in trusted hosts. One major problem of using such an approach is determining when or whether to add a host to, or revoke a host from, the list of trusted hosts.

2.4.3.1.2 Host reputation

Rasmusson et al. [140] suggest that agents could report hosts to a central registration agency which maintains trust assessments (reputations) of hosts, so that hosts behaving in unauthorised ways damage their reputation. Such an

evaluation process uses a social control approach, in which entities are judged according to defined behavioural rules. The problem with such an approach is that it is very difficult to make reputation systems robust against gaming by malicious hosts which know how reputation values are calculated. For example, a correctly performing host could be given a poor reputation score by a malicious agent, and a malicious host could behave correctly for a period, thereby gaining a 'good' score, before misbehaving.

2.4.3.2 Prevention techniques

2.4.3.2.1 Encrypted Functions

Sander and Tschudin [152–154] and Wilhelm [193] propose a technique involving executing an encrypted agent without decrypting it. The idea is to permit a host to execute an agent carrying an encrypted function without revealing the original function. The scheme operates as follows.

- Alice has a function f and computes the encrypted function $E(f)$.
- Alice sends a program $P(E(f))$ that executes $E(f)$ to Bob.
- Bob executes $P(E(f))$ using input x, and sends $P(E(f))(x)$ back to Alice.
- Alice 'decrypts' $P(E(f))(x)$ and obtains $P(f)(x)$ by some means not available to Bob.

It is noted that even if the function f is encrypted, a malicious host might still be able to mount a black-box attack, as discussed in §2.4.2.1.2. A host could repeatedly execute $P(E(f))$, and in certain circumstances might then be able to use the obtained pairs of inputs and (encrypted) outputs to reconstruct (reverse engineer) the function f. One solution to this problem proposed

in [4] involves use of a third party called a *secure computation service*, which is employed to execute the encrypted operations and which is trusted not to attempt to learn anything about the decrypted functions and their outputs. This technique is similar to the use of trusted execution environments, discussed in §2.4.3.2.4.

Barak [10] propose a somewhat similar technique designed to obfuscate the code so that it becomes difficult to analyse its functionality in real time. A more recent study of this technique can be found in [160].

Historically, whilst the notion of computing using encrypted data is very attractive, not least because of possible applications in the cloud, there is a shortage of practical schemes. However, in recent years a number of possible schemes have been proposed which come close to realising a general purpose function of the desired type [32,135]; recent work of Gentry [65] is of particular interest, although a truly practical scheme of universal applicability is not yet available.

2.4.3.2.2 Time-limited black boxes

An agent is called a *black box* if its data or code cannot be accessed and modified. Relaxing this definition a little, an agent is called a *time-limited black box* if it is a black box for a specified time interval. Hohl [76] proposed a scheme of this latter type that obfuscates the agent code in a manner that makes it hard to 'comprehend' the code semantics, and inserts an (unforgeable) expiration date into the code. However, the scheme has a number of practical limitations arising from the difficulty of successfully obfuscating the code (as discussed in §2.4.3.2.1), and the problem of choosing an appropriate block box time interval.

2.4.3.2.3 Environmental key generation

Riordan and Schneier [145] proposed a technique in which a 'clueless' agent is unaware of some of its possible future actions, because portions of its code or data are encrypted with an unknown key. After an agent's migration has commenced, the agent receives input from its environment which enables it to generate the secret key necessary to decrypt its code or data. Two variants of the scheme are proposed, namely the *forward-time approach*, which permits key generation only after a specific point in time, and the *backward-time approach*, which permits key generation only before some point in time.

A recent study of this technique can be found in [192]. One limitation of this scheme is that access to sensitive mobile code information will still need to be protected against the entities that might be involved in the key generation process.

2.4.3.2.4 Trusted execution environments

Agents can be protected by executing them on trusted tamper-proof hardware [194] or secure co-processors [199]. The use of software-based tamper-proof environments has also been proposed in [76, 189]. More recently, it has been suggested that agents can be securely executed in hosts using trusted computing technologies [72].

2.4.3.2.5 Agent spreading

Borselius et al. [19] suggest that the threats posed by a malicious host to trading agent transactions can be reduced by spreading a single trading task amongst multiple agents. The authors propose two possible ways of achieving this.

- In the first scheme, agents are equipped with 'shares' of the means to

commit to a transaction. They then jointly commit to the transaction using a threshold signature scheme, such as the scheme of Shoup [161].

- Alternatively, a trusted host can be used as the final destination for the agents, after they have visited a number of untrusted hosts. Following analysis of the agents, the trusted host can commit to what it believes to be the optimal transaction.

Ng and Cheung [127] propose a similar method in which mobile agents are divided into many independent parts, and computational tasks are distributed among multiple hosts.

2.4.3.3 Detection techniques

2.4.3.3.1 Replication

Schneider [157] and Yee [200] discuss the use of replication to detect attacks on mobile agents.

2.4.3.3.1.1 Replication of hosts

The host-replication approach of Schneider [157] works on the assumption that that there are enough agent replicas to ensure that some of them will escape attacks from, or encounters with, malicious hosts. Consider a case where a mobile agent executes in a sequence of *stage actions* S_i $(0 \leq i \leq n)$, and has an itinerary of hosts to visit. Stage S_i is processed at host i, and for each stage multiple hosts provide the same set of services. More formally, at stage i, there is a set $\{A_0^i, \ldots, A_n^i\}$ of hosts $(0 \leq i \leq n)$. Once an agent has been executed, each host A_k^i sends a copy of the agent to all the hosts implementing stage $i + 1$. Similarly, each host A_k^i receives multiple copies of the same agent. By comparing the received agents, a host can determine

whether or not no more than half of the hosts in stage $i-1$ are malicious. The host can then choose to execute the agent with the most frequent state. Details of a protocol that can be used to identify the malicious hosts in stage $i-1$ are given in [157].

One major problem with this approach is that it may be unrealistic to assume that an agent will only be attacked by a subset of the hosts implementing each stage. In addition, the scheme will not be appropriate in all scenarios. For example, replication is difficult to apply to an e-ticketing agent with purchasing capabilities.

2.4.3.3.1.2 Replication of agents

The agent-replication approach proposed by Yee [200] detects an attack on the e-ticketing scenario by comparing the results of two replicated mobile agents. These two agents follow the same predetermined itinerary but in reverse order, and the scheme works on the assumption that there is only a single malicious host in this itinerary. Suppose that the first agent encounters the malicious host at state i, and this malicious host modifies the best offer provided by a host at stage j of the route, $j < i$. However, the malicious host cannot modify the offer provided by the jth host to the second agent (with the reverse route). As a result, a comparison of the results produced by the two agents will yield the best offer, and will also enable the attack to be detected (but not the attacker).

2.4.3.3.2 Detecting black box attacks

Black box attacks (see §2.4.3.2) typically involve analysing how a black box agent reacts given a range of input parameters. Hohl and Rothermel [77] discuss how to detect black box attacks by making an agent maintain a log

of host responses to mobile agent requests. At each visited host the agent sends a message to a trusted host containing a unique identification number associated with the agent's code call, and a hash value of the host's data response. The trusted host can then compare the host response hashes for equal code calls. The trusted host will return an error to the agent if the host response hashes for equal calls do not match. One limitation of this approach is that a malicious host might not allow the agent to log host responses that reveal malicious host behaviour.

2.4.3.3.3 Detection using cryptography

Standard security services (such as authentication, confidentiality, integrity and accountability) can be offered with the use of standard cryptographic protocols, as discussed in §2.2.3. These mechanisms can help address attacks on the migration process of mobile agents, such as *impersonation attacks* (also known as *kidnapping host* attacks), which involve modifying the agent's owner Identification Data (ID) and signing the agent with a different private key. A malicious host could also attack a mobile agent by deleting data added by a previous host, as well as any corresponding tracing data (e.g. a digital signature).

Security protocols designed to address attacks on multi-hop migration routes are discussed in [104], where *append-only logs* are introduced; further proposals are given in [149]. Using such protocols, an attack can be identified after the mobile agent returns to its originator and the integrity of the host-inserted logs is checked. However, the proposed schemes are subject to a cut-and-paste attack in which two malicious hosts on the agent itinerary collude [149].

A similar, yet simpler, technique using a Partial Result Authentication Code (PRAC) has been proposed by Yee [200]. Rather than relying on asymmetric encryption of partial results, this technique uses symmetric encryption, which is faster to compute.

2.4.3.3.4 Forward integrity

Bellare [12] proposed a technique called *forward integrity* in which mobile agents produce partial logs as they proceed on a multi-host itinerary. This can be viewed as the integrity analogue of 'forward privacy' (also known in the literature as 'perfect forward secrecy').

The aim of this mechanism is to generate a MAC for each audit log in such a way that, even if the MAC key is compromised, data pertaining to the past cannot be forged, i.e. the attacker cannot modify entries or create non-genuine entries for past logging events.

2.4.3.3.5 Detection using cryptographic tracing

Vigna [184,186] proposed a technique called *execution tracing* that makes it possible to save the history of execution at a host in an unforgeable way. The protocol requires the nth host A_n, $n \geq 1$, visited by an agent to compute an execution trace for this agent. The trace is securely stored by A_n and is subsequently sent to a trusted host A_0. The protocol has two main components, which are summarised below.

Secure multi-hop migration. The host A_n, $n \geq 1$, supports execution of the mobile agent until the agent decides to migrate to the next host A_{n+1}. The host A_n then computes an execution trace for the agent, and sends this trace along using the agent to A_{n+1} with a protocol that supports

the required security services. This protocol uses hash functions and digital signatures to offer authentication, confidentiality and integrity (see, for example, §2.2.3).

Notification of migration. The host A_{n+1} receives the message from A_n and verifies that it was the intended recipient, that the mobile agent and its state were actually sent by A_n, and that the agent and state have not been modified. If so, then A_{n+1} sends A_0 a message that confirms receipt of the agent and execution trace. Further details are given in [186].

When the mobile agent completes its migration, the originator A_0 can check correct agent execution by simulating agent execution at all visited hosts using the traces it has received from each of these hosts. The trace consists of the values returned by all system calls made by the agent while executing on the host.

One limitation of this method is that the size of the trace grows linearly with the number of visited hosts. Vigna [185] proposes a number of techniques that can help reduce the tracing overheads. The security features of the original protocol were enhanced in [103] and [177]; however, the practicality of the approach remains questionable.

2.4.3.3.6 Detecting itinerary manipulations

Roth [148] proposes a technique that involves pairs of agents cooperating to detect agent migration blocking attacks. The two agents migrate and exchange information at each host they visit. At each host, one agent sends the other the names of the previous, current, and next host in its migration path. The receiving agent can examine this information for migration inconsistencies.

One drawback of this protocol is its communication overhead. Also the protocol assumes that the two agents are always executing on different hosts; a pair of colluding hosts could attack the protocol by, for example, killing both agents simultaneously.

2.4.3.3.7 Detecting truncation

In the previous sections we have discussed a number of techniques that can be used to detect attacks on agent data, code and itineraries. Some of these techniques can also be used to protect against truncation attacks (see §2.4.2.1.3).

A well-established method of detecting truncation attacks is the use of secure *chain relation* protocols. Karjoth et al. [103] introduced a chain relation protocol that uses digital signatures and hash functions to link the result obtained at the currently visited host to the result generated at the previously visited host and the identity of the next host to be visited. This protocol extends Yee's forward integrity using the PRAC scheme [200] (see §2.4.3.3.3). Similar protocols have also been proposed in [36,104]. However, none of these protocols protect against truncation attacks in which two or more hosts collude. This issue is addressed in [31], where a scheme is proposed in which previously visited hosts jointly sign a result generated at the current host; as a result, colluding hosts need previously visited hosts to jointly sign a fake offer as part of any attack. This work has been further developed by a number of authors [109, 114, 198, 202].

Other schemes protecting against truncation attacks use a TTP to record itinerary information [36, 79]. Such schemes, however, require a mobile agent to communicate with a TTP every time it visits a host, which introduces

a significant network overhead. However, this problem can be addressed by using trusted execution environments (see §2.4.3.2.4).

2.4.4 Host protection controls

We outline below host protection methods, designed to prevent agent technology from being used for malicious purposes.

2.4.4.1 Java security

The most widely discussed mobile agent toolkits use Java technology and security. A helpful introduction to Java security can be found in [130]. Java security functionality is particularly useful in protecting hosts against malicious agents, as discussed below.

2.4.4.1.1 Code signing

Java implements standard cryptographic functions to support agent integrity, confidentiality and authentication.

2.4.4.1.2 Bytecode verifier

The code of a Java class can be verified at the bytecode level. This enables the structural correctness of the Java class to be verified with respect to Java semantics.

2.4.4.1.3 Sandboxing

Sandboxing protects the VM during runtime in a number of ways. For example, *permissions* specify actions that code is allowed to perform within a *protection domain*, as determined by *policy files* defining the permissions given to code executed within the domain. Signed code can be verified with the aid of certificates contained in *keystores*.

2.4.4.1.4 Java security limitations

Java security is not a panacea; a properly signed agent code might still contain malicious content and harm a host. The security limitations of Java are discussed by Roth [150]: "*Java has simultaneously been a fortune and a misfortune. Without the many features of Java, it is undoubtedly more difficult to develop a mobile agent toolkit—on the other hand, all these limitations and shortcomings of Java with regard to security make it next to impossible to build and maintain a publicly deployed and dependable mobile agent system.*"

2.4.4.2 Agent trustworthiness

Code signing, discussed above, is used in almost all mobile agent toolkits to verify the origin and integrity of an agent and to authorise certain agent requests. We discuss below other techniques that can be used to verify the trustworthiness of agent code.

2.4.4.2.1 Proof-carrying code

This technique, proposed by Necula and Lee [125, 126], provides a host with guarantees that it is safe to execute an untrusted piece of mobile code. The host publishes a safety policy describing the properties which mobile code must satisfy. The agent must then contain a proof that its code complies with this safety policy. The proof-carrying code is a special form of intermediate code representation containing the mobile code and an encoding of a formal proof that the code complies with the safety policy. This approach is currently an active and promising research topic [3, 113].

2.4.4.2.2 Path histories

Trust in a network transaction typically depends on the identity of the

communications partner, which is verified in an authentication process. For a mobile agent, trust might also depend on the agent's path history [134]. A path history enables a host to verify the sequence of hosts an agent has previously visited. This might be important if, for example, a host maintains a list of trustworthiness scores of other hosts. This technique is similar to the trusted host itinerary and host reputation techniques discussed in §2.4.3.1.

2.4.4.2.3 State appraisal

Farmer et al. [58] proposed a technique that helps detect attacks on an agent's state. Unauthorised modifications to an agent state can be predicted, described, and later verified using a state appraisal function provided by the agent owner. Such a function defines a set of permissions that limit the functionality of hosts that the agent visits. The function will then check that these permissions have not been violated. For example, this might be verified by checking that certain conditionals are true, or that certain invariants have the expected value.

This approach can be combined with the concept of *detection objects*, i.e. 'bait' that can be inserted in agent code that does not affect the code's main functionality [116]. If a detection object is modified then it can be deduced that the code has been tampered with. This technique relies on the detection object being non-obvious; i.e. the presence of the detection object must be concealed from executing hosts.

The state appraisal technique has also been studied by Berkovits et al. [14]. Stengel et al. [171] propose a similar method for detecting mobile agent state manipulation by observing agent communication patterns, e.g. the repeated transmission of the same information.

2.4.4.2.4 History-based access control

This technique involves maintaining a selective history of accesses requested by the code, and using this information to distinguish between trusted and untrusted code [52]. It is claimed [52] that in some applications it may be too restrictive to use static privileges; e.g. if all accesses to the local file system were to be prohibited, then it would not be possible to open a socket to a remote host. Instead, mutually exclusive privileges could be used, e.g. the code could be allowed to read a file but not open a network connection afterwards (and vice versa). The scheme employs unique identifiers for programs, which are computed using hash functions. This prevents the approach being used for programs that use dynamic code loading during runtime.

2.4.4.2.5 Host architecture

Borselius [17] proposed a universal host architecture which can be used to describe mobile agent functionality in remote hosts. This architecture is depicted in Figure 2.3, and includes the following components.

- AEE: is the governing management and control function, responsible for all agents executing on the platform.
- Communication services: provide communications facilities to agents.
- Security services: provide agent security controls.
- Mobility service: enables and controls agent migration.
- Event logging service: logs security related events for storage in an audit trail.
- Security policy and access control database: regulates the behaviour of security mechanisms.

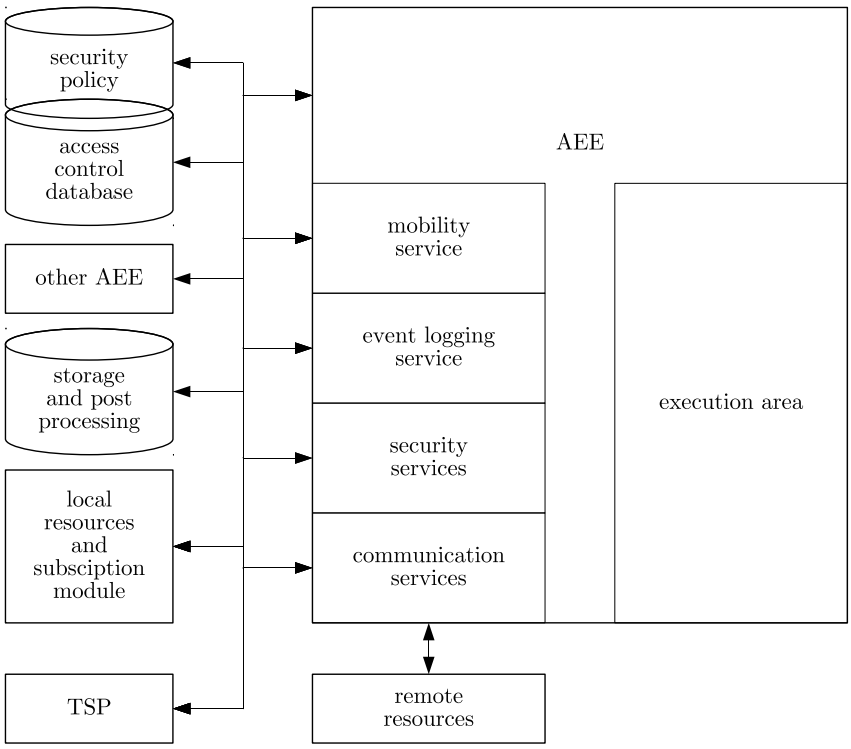


Figure 2.3: Agent execution environment architecture.

- Storage and post processing: manages and processes log data.
- Local resources and subscription module: provides access to resources (hardware or software).
- Trust Service Provider (TSP): provides various trust services.
- Remote resources: provides access to resources residing on other platforms.

2.4.5 Summary: mobile agent security controls

We have reviewed a range of security mechanisms protecting mobile agents and hosts. Host protection technologies have been developed to the point where some authors consider them adequate for some applications [24] (although there remain unresolved issues). On the other hand, there remains a lack of

Table 2.2: Mobile agent security solutions.

Countermeasure	Subject	Type	Code or data
Encrypted functions	Agent	Prevent	Both
Time-limited black boxes	Agent	Prevent	Both
Environmental key generation	Agent	Prevent	Both
Host replication	Agent	Detect	Both
Agent replication	Agent	Detect	Both
Detecting black-box attacks	Agent	Detect	Data
Static data integrity	Agent	Detect	Data
Forward integrity	Agent	Detect	Data
Execution tracing	Agent	Detect	Both
Secure itinerary recording	Agent	Detect	Data
Secure chain relation	Agent	Detect	Data
Sandboxing	Host	Prevent	Code
Code signing	Host	Detect	Code
Proof-carrying code	Host	Prevent	Both
Path histories	Host	Detect	Both
State appraisal	Host	Detect	Data
History-based access control	Host	Prevent	N/A

completely effective security measures to protect mobile agents against malicious hosts [57]. Table 2.2 summarises the mobile agent security techniques that have been discussed in the previous sections, [22]. Further information can also be found in the outputs of the Mobile VCE project[6].

It should be emphasised that the overhead (communication or computational) associated with certain mobile agent security measures needs to be carefully considered before adoption. For example, forcing a mobile agent at each host to securely communicate tracing logs with a trusted party (resulting in a star-shaped communication pattern) may defeat the purpose of using mobile agents to reduce the communication and computational overheads arising from the use of standard client-server architectures.

☞ This thesis focuses on mobile agent security; the objective is to use spy

[6] http://www.mobilevce.org

agents to identify the origin of a attack, based on information obtained using attack detection techniques.

2.5 Other security topics

2.5.1 Data privacy

2.5.1.1 Notions

This section provides background on data privacy. We first give the following fundamental definition.

Definition 2.8 (ISO 7498–2, [83])**.** *Privacy is the right of individuals to control or influence what information related to them may be collected and stored by whom, and to whom that information may be disclosed.*

An alternative, somewhat more specific, definition is given by the Organisation for Economic Co-operation and Development (OECD):

Definition 2.9 (OECD Glossary of Statistical Terms, [132])**.** *Privacy is the status accorded to data which has been agreed upon between the person or organisation furnishing the data and the organisation receiving it and which describes the degree of protection which will be provided.*

A more functional definition of privacy is given in ISO/IEC 15408:2008, which covers evaluation criteria for information technology security and, in particular, security functional components [84]. This multipart standard is also known as the *Common Criteria* (for Information Technology Security Evaluation). It expresses privacy as a 'functional class' with four 'member families': *anonymity*, *pseudonymity*, *unlinkability*, and *unobservability*. Definitions for these families have also been provided by Pfitzmann and Hansen [138]

with the purpose of establishing a consistent terminology within the research community.

Providing a privacy service often relies on other security services such as authentication, confidentiality, accountability, and access control. For example, the use of cryptographic mechanisms can ensure that (personal) data is controlled by, and is only accessible to, authorised entities, and that these entities are accountable for the exercise of such privileges. However, privacy services might also conflict with security services. For example, the provision of accountability could be adversely affected if anonymity is used to protect user privacy.

2.5.1.2 Regulations

Data privacy is the subject of a wide variety of legislation. For example, in Europe privacy is addressed by the European Union (EU) Data Protection Directive (aka 'the Directive') [164]. The Directive defines 'personal data' as any information relating to an identified or identifiable natural person ('data subject'); an identifiable person is "one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity" [164, article 2(a)]. The Directive requires that personal data should (among other things) a) "be collected for specified purposes and not be further processed for other purposes", and b) "be merely adequate and not excessive for the purposes motivating its collection" [164, article 6(1)].

The Directive helps to protect against inadvertent disclosure of private data to unauthorised parties. However, its scope is restricted to safeguard

other (conflicting) interests such as a) national or public security; b) police investigations; c) important economic or financial interests; and d) monitoring, inspection or regulatory functions connected, even occasionally, with the exercise of official authority in previous cases. These exceptions (some of which are equivocal) weaken the protection it provides.

Privacy can also be protected by requiring domains in which private data is exposed to abide by particular data protection policies. These policies may be based on standard *privacy principles* such as notice, choice and consent, collection, use and retention, access, disclosure to third parties, security for privacy, quality, and monitoring and enforcement (see, for example, the OECD Guidance on Policy and Practice, [131], or the UK Data Protection Act, [182]). In this direction, the Platform for Privacy Preferences (P3P) [141] defines a framework allowing a policy to be negotiated by two parties. For example, during a registration process, a user may be requested to read the policy and opt in if they agree with it [102]. However, such *notice and consent* frameworks have been widely criticised and should not be relied upon for privacy protection [78]. For example, even though individuals may wish to protect their privacy, they may still disclose personal information for reasons of convenience, financial incentives, and/or a lack of understanding of the consequences.

2.5.1.3 Detection and prevention

Breaching a data privacy policy is a security threat posed by hosts that handle mobile agents, particularly if the agents are issued by individuals. Personal email addresses provide an example of private data which might be held by a mobile agent whose use could be protected by a privacy policy. Unauthorised

use of an email address has a number of possible (detectable) side effects, including the reception of unsolicited messages (spam), identity theft, phishing, viruses, worms, malware and adware.

Information that can be used to uniquely identify, contact, or locate a single individual is known as Personally Identifiable Information (PII). Although data privacy is heavily regulated, the growing trend towards mass collection and exploitation of PII (e.g. data mining, data retention) in databases, renders regulations and security policies hard to enforce. This becomes even more challenging in mobile agent scenarios in which PII is willingly exposed to multiple hosts, within the scope of data protection agreements. The misuse of PII contained in mobile agents is a passive attack, which leaves no signs of violation in the mobile agents and may only be detected indirectly (e.g. by checking an email address for unauthorised use).

Even though privacy attacks can be detected (e.g. by employing e-mail filters), it might still be hard, or even impossible, to identify the origin of such attacks. For example, spammers might use open proxies to remain anonymous during harvesting and spamming. More specifically, spammers might collect private data using protocol parsers (potentially both at a low level, e.g. IP traffic monitors, and at a higher level, e.g. chattering); administration monitoring procedures (e.g. Internet Service Provider (ISP) procedures); or higher level applications (e.g. email marketing, newsgroups, Web browser plug-ins leaking information, or other spyware) [6]. Further, spammers and *blackhats* might use third-party relays or compromised hosts (e.g. *botnets*) in order to route large volumes of e-mail messages [91].

Privacy attacks can be prevented by employing PETs [7, 23, 73]. Examples of such technologies include the use of *blind signatures* [1], *mix networks* [29],

crowd systems [144], and *privacy assurance seals* [13, 122, 168, 195].

☞ Spy agents can help identify hosts that are responsible for email privacy infringements; this is discussed in greater detail in Chapter 10.

2.5.2 Monitoring agents

The notion of spy agents and mobile agent preemptive security assessments, discussed in this thesis, can be seen as techniques for providing internet security monitoring services. Strategically placed control mechanisms can perform service monitoring, [26,67], including monitoring of enhanced IP services, Quality of Service (QoS) and VPN packets. Similarly, in distributed intrusion detection systems, [28, 165], control agents can be used to protect a domain by 'interrogating' suspicious agents.

Mobile agents can be used to to assist in intrusion detection; for example, agent requests and delegation rights could be assessed by a trusted 'security management component' that is concerned with the security of the host and its execution environments [183]. As compared to the use of single-point security systems (e.g. *firewalls*), agents in a distributed system might be able to handle detected intruders more efficiently. Similar agent protection systems have also been proposed for wireless ad hoc networks [95, 179, 201]; the systems involve analysing audit data collected by wireless network agent sensors both to detect intrusions and to deter intruders. Detection of misbehaviours can be performed using a range of methods; for example, models have been proposed that emulate natural immune systems, or evolve using machine learning and fuzzy logic algorithms [40, 43, 92].

2.5.3 Deception

Deception in the form of social engineering and code breaking is commonly used by hackers [121]. However, it has been observed that it can also be used for attack detection and prevention by providing the attacker with a false reality (fiction) [108]. Indeed, deception is one of the main requirements for effective operation of a spy agent system.

Deception can be *consistent* or *inconsistent*. In consistent deception the objective is to ensure that the attacker does not perceive that deception is used. In contrast, in inconsistent deception the objective is simply to confuse the attacker, regardless whether or not the attacker perceives an act of deception.

We next describe two security protection systems that are based on the notion of deception.

2.5.3.1 Honeypots

The origin of remote attacks can potentially be identified with the aid of 'honeynet' systems [139], which "are nothing more than a security resource whose value lies in being probed, attacked, or compromised" [166]. As such, honeynets (aka honeypots) are designed to resemble valid systems; they use this masquerade to collect information about attackers and their methods.

Honeypots can be used in various ways. For example, email address harvesting and spamming may be detected in the following ways [5].

- A honeypot installed in a web site could 'poison' harvesters with decoy email addresses.
- A honeypot installed on an *open proxy* could discover a spammer's identity.

- A honeypot installed on an *open relay* could detect and block unsolicited emails.

The objective of a honeypot system is to collect valuable information from unsuspecting attackers. This is not always possible; for example, an attacker could use anti-honeypot techniques to detect the existance of a honeypot. For example, the *Send-Safe's Honeypot Hunter* "is a tool designed for checking lists of HTTPS and SOCKS proxies for so called 'honey pots'. 'Honey pots' are fake proxies run by the people who are attempting to frame bulkers by using those fake proxies for logging traffic through them and then send complaints to ones' ISPs[7]."

If a honeypot can be detected, it can potentially also be attacked. For example, a malicious user might be able to flood (poison) a honeypot with false information [106]. By poisoning the honeypot, a malicious entity's other hostile activities might go unnoticed. More generally, just as spam originators evolve new ways of bypassing spam filters, spammers are also developing ways of combating honeypots. Hence honeypots can only continue to be a useful security mechanism if their operators maintain technical superiority over the adversaries of the resources being protected.

2.5.3.2 Software decoys

A decoy is intended to deceive something or someone into believing it is the object it advertises itself to be. Therefore, the creator of a decoy must make the decoy resemble a genuine object as much as possible in order to effect the desired deception. Daniel and Herbig define deception as the "deliberate misrepresentation of reality done to gain a competitive advantage" [39].

[7] http://www.send-safe.com/honeypot-hunter.php

Intelligent software decoys bear certain similarities to honeypot systems; however, they differ from honeypots in a number of ways. Decoys are used to actively defend components rather than just collect data; they are not designed solely to attract the attention of adversaries; and they do not rely on the concept of a *perimeter of defences* (in particular they might operate autonomously) [118].

☞ One of the aims of deception-based security mechanisms is to deter parties from performing malicious acts. As compared with decoys and honeypots, spy agents differ in that attacks on mobile code take place in an adversary's environment, rather than a trusted one.

2.6 Combinatorial group testing

In this thesis we use combinatorial group testing (GT) to construct spy agent routes. We next provide a short introduction to this mathematical theory.

2.6.1 Introduction to group testing theory

In GT a (large) population of items containing a small set of *defectives* is tested in order to identify the defectives [45]. Items can be pooled together for testing; a *group test* reports 'positive' if the tested pool contains one or more defective elements, and reports 'negative' otherwise.

There are two main types of GT algorithms: sequential and nonadaptive. Sequential algorithms allow the selection of later tests to be based upon the outcomes of previous tests. In a non-adaptive scheme, the set of tests is completely predetermined. Sequential algorithms require, in general, fewer tests, since the extra information allows for more efficient designs. On the

other hand, non-adaptive algorithms allow tests to be conducted in parallel which, in general, reduces the overall test time if not the number of tests.

The GT problem is often denoted by $S(\overline{d}, n)$ where S is the sample space of objects to be tested, where at most d defective items from the sample space are to be detected. If it is assumed that the sample space contains exactly d defectives, then the GT problem is denoted by $S(d, n)$.

2.6.2 Sequential group testing

The history of Sequential Group Testing (SGT) algorithms goes back over 60 years [44]. There are two main mathematical models for the classical GT problem [45].

- Probabilistic models assume that a certain item is defective with probability p (typically using a Bernoulli distribution).
- Combinatorial models assume that the maximum number of defective items is known prior to testing.

In the original GT work [44,163] the problem was to determine which blood samples from a certain population contain the syphilis antigen. When individual blood sera were pooled in groups, the test being used had the property that:

- if none of the sera in the pool contained the syphilitic antigen, then the pool tested 'negative', and
- if one or more of the sera in the pool contained the syphilitic antigen, then the pool tested 'positive'.

The goal of the GT design problem as originally formulated was to detect all the positive samples using the minimum number of tests (as a matter of

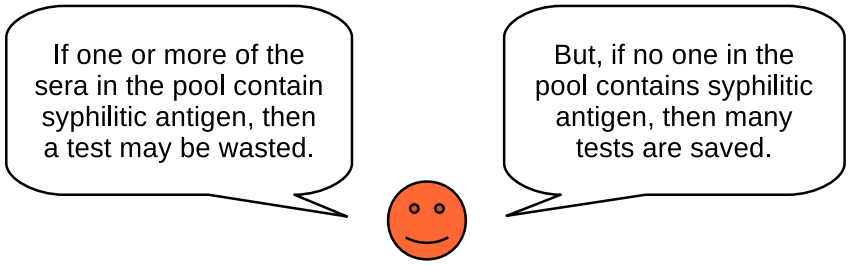


Figure 2.4: The idea of group testing.

economy). This optimisation problem is summarised in Figure 2.4.

Dorfman [44] showed that, when considering a Bernoulli probability distribution and a small p, the best group size S is approximately $p^{-1/2}$ and the resulting minimal ratio of tests to items tested is approximately $2p^{1/2}$.

In SGT, a fundamental objective is to minimise the total number of tests required to find all the (up to) d defective items within a group of n items.

For given values of d and n, we therefore write $N(d, n)$ for the minimum number of tests in a SGT scheme which can identify up to d malicious hosts from a set of n hosts. Since each test divides the sample space of hosts $S(d, n)$ into two disjoint sets, the following information theory bound applies [37, Section 5.4: Bounds on the Optimal Code Length]: $N(d, n) \geq \lceil \log_2 n \rceil$. In general, it is not easy to determine $N(d, n)$. In fact, Du and Hwang [45] proved that, if $d = 1$, then $N(1, n) = \lceil \log_2 n \rceil$, and this is the only case in which the exact value of $N(d, n)$ is known [45]. General purpose SGT algorithms include Hwang's generalised binary splitting algorithm [93] and Li's s-stage algorithm [110].

2.6.3 Non-adaptive group testing

Non-adaptive algorithms have been studied for a shorter time: Non-adaptive Group Testing (NGT) theory first arose in the study of superimposed codes

[105]; however the connection with NGT was only established much later in the study of the d-complete designs [25] and the hypergeometric NGT problem [82].

Much recent NGT research has been driven by applications in Deoxyribonucleic Acid (DNA) library screening in molecular biology, in which the items are DNA subsequences (clones) and tests are performed on pools of clones to determine which contain a particular DNA sequence [128]. In this context NGT schemes are also known as *pooling designs* [8].

In the *hypergeometric* problem, it is also assumed the number of defectives never exceeds d; in the *strict* problem, it is required to verify this assumption. That is, a scheme will only identify a set of defective items if the assumption has been verified.

The following tentative taxonomy of non-adaptive pooling designs is given in [128].

- Deterministic designs: Every pool is deterministically specified.
- Random designs: Some or all of the entries are randomly determined with parameterised probabilities that can be optimised based on certain criteria.
- Error-tolerance designs: Deterministic or random designs with the additional ability to tolerate tests yielding erroneous outcomes (e.g. reporting 'false positives').

A range of NGT constructions have been proposed based on block designs [35], superimposed codes [49, 50, 105], transversal designs (e.g. grid designs [11]), cover-free families [54, 55, 175, 190], and other combinatorial designs [9, 11, 54, 111, 129] (see §2.6.5). A recent survey of NGT can be found

in [46].

2.6.4 Group testing for complexes

In (classic) GT a set of elements possible containing defective elements is tested in order to identify any defective elements. A test on a subset of elements gives a positive result if and only if this subset contains at least one defective (or *positive*) element. An important extension to the standard GT theory is the theory of Group Testing for Complexes (GTC), in which elements can be collectively positive. We call such a collectively positive element a 'defective complex'. In this model, a test gives a positive result if and only if the test set contains at least one defective complex.

GTC is commonly associated with *hypergraph testing*. A hypergraph consists of a set of vertices (corresponding to elements to be tested) and a set of edges (i.e. sets of vertices) that correspond to the (candidate defective) complexes. The rank of an edge is the number of vertices in it. In this representation Du and Hwang define the following particularly important hypergraphs [46, Chapter 6]:

- r-graph: a hypergraph whose edges all have rank r,
- $\overline{r}$-graph: a hypergraph whose maximum rank is r,
- r^*-graph: the complete r-graph.

In hypergraph testing [63], the problem is to identify a hidden subgraph using a small number of tests. This problem was first studied in the context of identifying DNA complexes [112], in which there is a set S of molecules and an unknown family $\mathcal{D} = \{D_i\}$ of subsets of S, where each subset is a cause of a certain disease. Hence, unlike in NGT, in hypergraph testing a combination

of a set of elements is required to induce a positive effect. This defines the fundamental GTC problem, where we wish to identify the 'set of defective complexes', $\mathcal{D}$, using a small number of tests.

Chen and Wei [30] showed that the GTC problem is connected to the secure Key Distribution Pattern (KDP) problem [120] (see §2.6.5.7), generalised superimposed codes [49] (see §2.6.5.5), and Cover-free Families (CFF) [175, 190] (see §2.6.5.4).

2.6.5 Useful combinatorial structures

2.6.5.1 Block designs

Combinatorial designs (block designs) are examples of set systems. The definitions below are taken from Colbourn and Dinitz [35]; standard set theory and notation can be found in [90].

Definition 2.10. *A Balanced Incomplete Block Design (BIBD) is a pair* $(V, \mathcal{B})$*, where* V *is a* v*-set and* $\mathcal{B}$ *is a collection of* b k*-subsets of* V *(*blocks*) such that each element of* V *is contained in exactly* r *blocks and any 2-subset of* V *is contained in exactly* λ *blocks. The numbers* v, b, r, k, λ *are the* parameters of the BIBD.

Definition 2.11. *The* incidence matrix *of a BIBD* $(V, \mathcal{B})$ *with parameters* v, b, r, k, λ *is a* $v \times b$ *matrix* $A = (a_{ij})$*, in which* $a_{ij} = 1$ *when the* i*th element of* V *occurs in the* j*th block of* $\mathcal{B}$*, and* $a_{ij} = 0$ *otherwise.*

Definition 2.12. *A* t*-*(v, k, λ) *block design (or simply a* t*-design) is a pair* $(V, \mathcal{B})$ *where* V *is a* v*-set of* points *and* $\mathcal{B}$ *is a collection of* k*-subsets of* V *(*blocks*) such that every* t*-subset of* V *is contained in exactly* λ *blocks.*

Remark 2.13. *A 2-*(v, k, λ) *design is a Balanced Incomplete Block Design (BIBD).*

For a 2-design the following equations apply [15]:

$$bk = vr \tag{2.6.1a}$$

$$\lambda(v-1) = r(k-1) \tag{2.6.1b}$$

Beth et al. [15] provide a thorough introduction to design theory; a comprehensive collection of results on combinatorial designs is given in Colbourn and Dinitz [35].

2.6.5.2 Vectors

Let $x = [x_0, x_1, \ldots, x_{n-1}]^T$ and $y = [y_0, y_1, \ldots, y_{n-1}]^T$ be binary n-dimensional vectors. The *superposition sum*, denoted by $x \vee y = [x_0 \vee y_0, x_1 \vee y_1, \ldots, x_{n-1} \vee y_{n-1}]^T$, is the element-wise logical OR of the two vectors. We say that a vector x is contained in a vector y, if $x \vee y = y$.

2.6.5.3 Separating matrices

A $u \times b$ binary matrix M with columns indexed by $\{1, \ldots, b\}$ is a:

- *d-disjunct matrix* if the superposition sum of any d columns of M does not contain any other column of M;
- *d-separable matrix* if, for every $D_1, D_2 \subseteq \{1, \ldots, b\}$ with $|D_1| = |D_2| = d$, the superposition sum of the columns indexed by D_1 and the superposition sum of the columns indexed by D_2 are equal only if $D_1 = D_2$;
- $\overline{d}$*-separable matrix* if, for every $D_1, D_2 \subseteq \{1, \ldots, b\}$ with $|D_1| \leq d$ and $|D_2| \leq d$, the superposition sum of the columns indexed by D_1 and the superposition sum of the columns indexed by D_2 are equal only if $D_1 = D_2$.

A family of v subsets $C_1, \ldots C_v$ of a b-set X is *d-disjunct* if, for every subset D of X of size at most d and for every $x \in X - D$, there exists an i $(1 \leq i \leq v)$ such that $x \in C_i$ and $C_i \cap D = \emptyset$.

Theorem 2.14. (Colbourn and Dinitz [35, Section VI.56]) *Every d-disjunct matrix is $\overline{d}$-separable. Every $\overline{d+1}$-separable matrix is d-disjunct.*

2.6.5.4 Cover-free families

A family of b subsets $B_1, \ldots B_b$ of a v-set V is [190]:

- *d-cover-free* if, for every $D \subseteq \{1, \ldots, b\}$ with $|D| = d$ and $i \neq D$, $B_i \nsubseteq \bigcup_{j \in D} B_j$;
- *d-weakly union-free* if, for every $D_1, D_2 \subseteq \{1, \ldots, m\}$ with $|D_1| = |D_2| = d$, $\bigcup_{j \in D_1} B_j = \bigcup_{J \in D_2} B_j$ implies $D_1 = D_2$;
- *d-(strongly) union-free* if, for every $D_1, D_2, \subseteq \{1, \ldots, m\}$ with $|D_1| \leq d$ and $|D_2| \leq d$, $\bigcup_{j \in D_1} B_j = \bigcup_{J \in D_2} B_j$ implies $D_1 = D_2$.

The notion of a d-cover-free family can be generalised in the following way [47]: A family of sets $\mathcal{B}$ is said to be (w, d)-*cover-free* if, whenever $\mathcal{B}_1, \mathcal{B}_2 \subset \mathcal{B}, |\mathcal{B}_1| = w, |\mathcal{B}_2| = d$, and $\mathcal{B}_1 \cap \mathcal{B}_2 = \emptyset$, it holds that $\bigcap_{B \in \mathcal{B}_1} B \nsubseteq \bigcup_{B \in \mathcal{B}_2} B$. A d-cover-free family is $(1, d)$-cover-free.

2.6.5.5 Superimposed codes

A *d-superimposed code* of *size* b and *length* v is a collection of b binary vectors (codewords) of length v, with the property that the superposition sum of a set D of d or fewer codewords uniquely determines D [49].

Theorem 2.15. (Colbourn and Dinitz [35, Section VI.56]) *d-Superimposed codes of length v and size b, $\overline{d}$-separable $v \times b$ matrices, and d-(strongly)-union-free families of b subsets of a set of size v are all equivalent.*

A superimposed code has *constant weight* w if every codeword contains exactly w ones. Equivalently, the corresponding set system is said to be w-uniform.

2.6.5.6 Frameproof codes

An electronic object (e.g. a file) can be *fingerprinted* by placing marks in locations within the object that do not affect its meaning and/or functionality.

Suppose that an object (e.g. a plaintext message) contains L possible locations for the inclusion of marks. If s values can be assigned to each of the marks, then we define a fingerprint as the sequence of values assigned to each of the L marks. Thus a fingerprint can be thought of as a word of length L over an alphabet Σ of size s. The process of fingerprinting an object then involves assigning a unique codeword from Σ^L to each user. By colluding, users can detect the location of a specific mark if the value assigned to it differs between their copies; otherwise, a mark location cannot be detected.

The main property that the marks should satisfy is that users cannot change the state of an undetected mark without rendering the object useless. Naive redistribution occurs when a user redistributes his copy of the object without altering it. If an unauthorised copy of the object is found containing user's u codeword, then user u is said to be guilty.

However, u could claim that he was framed by a coalition of users that created an object containing his codeword. We would therefore like to construct codes that satisfy the property that no coalition (of at most d users) can collude to frame a user not in the coalition. Such codes are called d-frameproof codes [16].

Finally, if a coalition colludes to generate an unregistered object, we would

then like (when this object is found) to be able to determine the users (or a subset of them) that colluded to create the object. Codes that support such traceability algorithms are said to be secure-frameproof.

Theorem 2.16. (Stinson et al. [172]) *A d-frameproof code is equivalent to a d-cover-free family.*

2.6.5.7 Key distribution patterns

Suppose that n users want to communicate securely in groups of size e. If a conventional (symmetric or secret key) cryptosystem is being used, where a trusted authority generates and distributes a secret key to each e-set of users, then $\binom{n}{e}$ keys would be required, and each user would need to store $\binom{n-1}{e-1}$ keys. The number of keys to be stored can be considerably reduced by somewhat weakening the security requirements. Suppose each user is equipped with a set of secret keys. Given a subset of e members and a disjoint subset of d non-members, we require that there exists a key owned by each member of the e-subset and by no member of the d-subset [120].

2.6.5.8 Some applications

GT designs and related combinatorial constructions (see above) have been used in addressing a range of information security problems, including:

- frameproof codes and traceability schemes [16, 33, 62, 169, 172–175];
- broadcast encryption [42, 64, 107, 174];
- key storage [51, 120]; and
- multi-receiver authentication [151].

Also, such combinatorial designs have been used in many other fields, including multi-user communications, in which contemporaneous messages must be successfully decoded [197], and DNA testing, where finding complex diseases needs to be performed in an economical way (see §2.6.4).

2.7 Conclusions

This chapter has reviewed a range of background topics. It has provided an introduction to cryptographic primitives, mobile agents, mobile agent security, data privacy, deception, and the theory of combinatorial group testing. This material provides the basis for the remaining chapters of this thesis.

"Long is the road from conception to completion."

Moliere

3 Spy agents

Contents

3.1 Synopsis

In this chapter we introduce spy agents and provide a spy agent system architecture and design methodology. More specifically, we describe how spy agents can be deployed within a variety of network protocol architectures in

order to perform high fidelity trust assessments of remote agent hosts. The spy agent framework consists of: a spy agent architecture that instantiates spy agents with appropriate content; a spy agent routing framework that dictates how spy agents are deployed; and an evaluation entity that implements the necessary security analysis mechanisms.

Spy agents is a novel concept; most related prior art falls within the area of mobile agent security (§2.4), while the closest (albeit different) concepts are those of agent spreading (§2.4.3.2.5) and replication (§2.4.3.3.1).

The rest of this chapter is organised as follows: the spy agent concept is discussed in §3.2 and §3.3, and the elements of the spy agent architecture are given in §3.4.

Most of the material in this chapter has been published in [101].

3.2 Introduction to spy agents

Spy agents, the notion of which was introduced in §1.3, are mobile software agents that are despatched from one (or more) secure platform(s) and migrate through a series of potentially insecure remote hosts, before returning to a secure location. Their purpose is to gather information to help evaluate the trustworthiness of visited platforms. Unlike malicious agents such as viruses, spy agents are legitimate mobile agents in the sense that they interact with visited hosts in the way expected by the hosts. As such they are analogous to software decoys (see §2.5.3.2) and honeypots (see §2.5.3.1).

The main objective of a spy agent system is to evaluate trust in remote environments, where the remote hosts should be unaware that their trustworthiness is being assessed. The structure of a spy agent should be no different to that of a generic mobile agent (e.g. an agent designed to obtain the prices of

goods from multiple e-commerce sites), as discussed in §2.3.4. A spy agent will include decoy private data and a pre-coded routing scheme, which hosts are typically able to access to enable migration. The evaluation of remote platforms is based on the collective outcomes of the spy agents (i.e. the detectable impacts on the agents after visiting the defined set of platforms). Each individual agent outcome could either be negative (if there is no sign of security violation) or positive (otherwise).

Spy agents can be used to mitigate the threats posed by malicious hosts by preemptively assessing the trustworthiness of remote hosts. Such assessments should take place before remote hosts are sent vulnerable mobile agents or sensitive data.

3.3 Developing the spy agent concept

The main idea behind a spy agent is to provide a means to evaluate trust in remote environments without the target hosts knowing that they are being assessed. This is achieved by selecting spy agent routes in such a way that malicious remote hosts will not suspect that they can be held responsible for any malicious acts. As a result, spy agents have the potential to determine the target hosts' genuine behaviour, i.e. the degree to which a host complies with its policies or, more specifically, with its responsibility to respect client security and privacy requirements.

A significant part of a spy agent's task is the extraction of security-related information from a remote host without violating the host's policy or security protection mechanisms. This means that a spy agent should retrieve the information it needs from remote hosts in a legitimate way and with the explicit permission of the host concerned; at the same time this must happen in a way

that does not reveal the spy agent's true objectives. The information retrieved by a spy agent will not necessarily be directly security-related; it simply needs to be information that can be used in some way to assess host behaviour, possibly when combined with information retrieved by one or more other such agents.

We identify the following fundamental principles underlying the use of spy agents.

- Target hosts should be incapable of deciding whether they are dealing with a spying scenario or not.
- Spy agents should appear as 'normal' (e.g. m-commerce) agents.
- Target hosts should be given a motive to misbehave by using spy agents as 'bait'.
- The results obtained from the dissemination of spy agents should be analysed in a safe environment.

As discussed above, spy agents are unlike viruses and other 'illegitimate' agents in that they should not breach host security and privacy. Instead, spy agents should interact with hosts in ways which hosts expect e.g. by complying with the host's 'sandbox' functional requirements. This concept is depicted in Figure 3.1.

We assume that the information acquired by spy agents is willingly provided by the hosts, directly or indirectly (e.g. by the impact of host actions), and that the use of such information for evaluation purposes is both legitimate and 'ethical'. Note that similar assumptions cannot be made about host behaviour, as mobile code is at the mercy of the host which executes it.

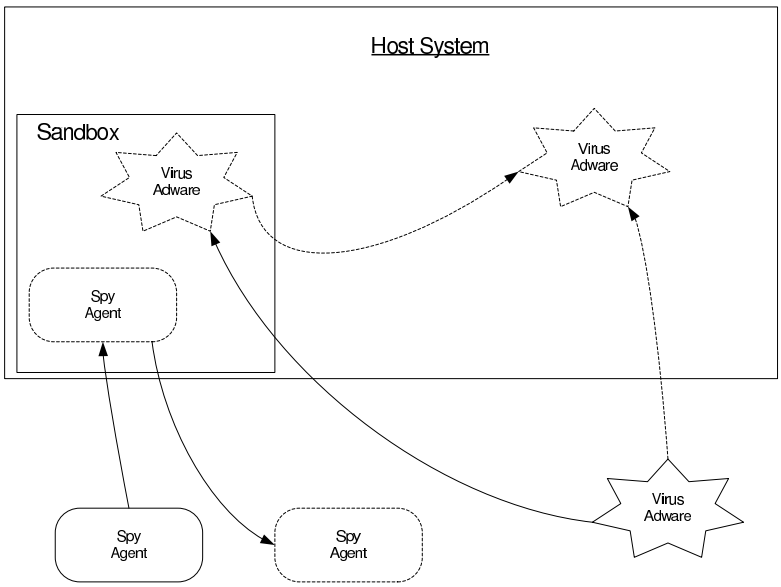


Figure 3.1: Spy agents versus malware and viruses.

☞ In summary, spy agents can be defined as legitimate mobile agents which are able to interact with remote, potentially hostile, mobile agent hosts in a manner that is expected by the hosts, and that support trust assessments without host knowledge.

3.4 Spy agent system architecture

A spy agent system involves the dissemination of a number of spy agents to target network nodes (hosts) in a network, and the retrieval of these agents following interaction with the nodes. As further discussed in Chapter 5, the deployment of a large number of spy agents can provide cross-referenced analyses of host trustworthiness.

The spy agent system can be regarded as an extension of a standard mobile agent system. We suppose that the spy agent system is based on standard mobile agent components, interfaces, and functions, including standard communication, mobility, and security protocols, as discussed in §2.3. Spy agents

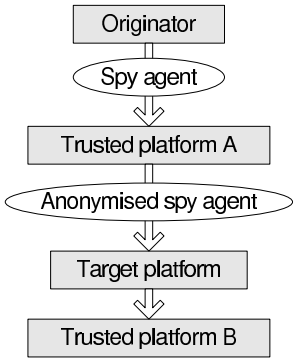


Figure 3.2: Fundamental spy agent system components.

can be implemented within any mobile agent framework, such as a system conforming to the FIPA specifications (see §2.3.3).

The rest of this section is concerned with architectural extensions to a mobile agent system that are required by a spy agent system.

3.4.1 Spy agent network components

A fundamental requirement is to provide spy agents with anonymity. A degree of anonymity can be achieved by associating a set of agents with a number of different sources or transmitters. That is, spy agents can be forwarded (by their originator) to a multiplicity of trusted nodes in the network. Each such node modifies the received agent's code in order to show itself as the source of the agent, before forwarding the agent towards the target node. Such an arrangement is shown in Figure 3.2.

For reasons that are analysed in more detail below, a spy agent should give away as little information about its purpose as possible. In this context it is proposed that the modified spy agent should ultimately be destined for a trusted node different to the node that sends it to the target host. This second trusted node will be notified by the first trusted node to expect a particular spy agent, and on receiving the agent will be able to forward it back to its

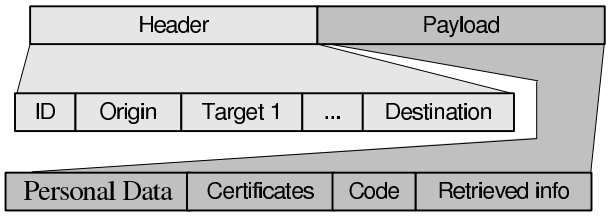


Figure 3.3: Spy agent internal structure.

origin.

When all spy agents return to their original source, the system can then analyse the agent interactions with the target hosts, in order to assess their level of trustworthiness.

3.4.2 Spy agent content framework

Spy agents will contain pre-coded routing and remote execution mobile code, and they should implement standard security protocols for access control, non-repudiation and data encryption (see §2.4).

A schematic of a software spy agent is shown in Figure 3.3. The agent includes an agent ID, an origin or source ID field, a final destination ID field, a number of intermediate node IDs, and a payload. The payload includes personal data such as a name, address, email address, digital certificates, security logs, financial information, and other information associated with a person or client. Finally, the agent includes executable code.

The spy agent's private data, such as ID information, email address, public key certificates, etc., should preferably correspond to a temporary entity set up by a mobile platform in a legitimate manner. That is, spy agents should appear to be originated by a normal client, and their relationship with the real client should be hidden. This helps to ensure that the target hosts will process the spy agent in exactly the same way as they would treat any other mobile

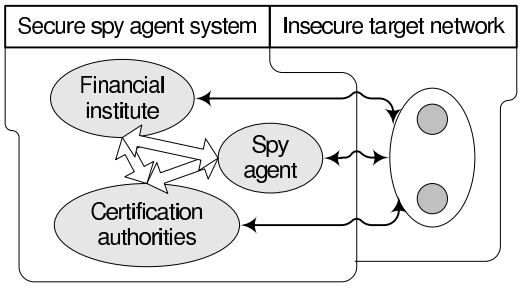


Figure 3.4: Spy agent system—interactions with trusted parties.

(e-commerce) agent. In order to be able to disguise a spy agent in this way, the spy agent system might need to (securely) interact with on-line services such as e-mail providers, certification authorities and banks, who would need to be aware of the purposes of the spy agent and be prepared to support them. These relationships are shown in Figure 3.4.

For example, the spy agent originator might need to set up a temporary email address or request a certificate from a certification authority for temporary use in assessing a host. This certificate need not allow an agent to perform any transaction automatically since it will be temporary. However a target platform should not be aware of this, and should believe that the agent will be equipped with all the 'normal' functions of an (m-commerce) agent. That is, it must appear to be just another mobile agent that could, if it wishes, decide to complete an electronic transaction.

The requirement for a spy agent to have a commonly used structure serves both the principle of mobile agent interoperability and the spy agent principles discussed in §3.3.

3.4.3 Spy agent routing framework

One of the most important aspects of a spy agent system is the routing mechanism. This determines the migration logic of each spy agent, and also influences the content of an agent, as described in the previous paragraph. The routing logic needs to address the complexity of dealing with multiple agents, issued from a multiplicity of trusted nodes, and routed over a variety of different paths. This complexity is necessary since the agent routing strategy plays a critical role in determining how well the system requirements are met. Amongst other things, the routing scheme should help to disguise the fact that spy agents originate from a specific client device and are in any way related.

A typical routing scenario is shown in Figure 3.5. A number of spy agents are created by the originator, anonymised by certain trusted platforms, routed via a number of target platforms, and are programmed to return to specified trusted platforms. Note that a trusted platform could, for example, be a mobile terminal, a home computer, or a public server set up for this purpose, although the latter option might increase network and end user costs.

In this thesis we assume that all the spy agent functionality in the trusted network (e.g. secure communication and anonymisation mechanisms) can be implemented using standard mechanisms. Following from this assumption, we focus on routing protocols within the (insecure) domain of remote target platforms. In line with the principles given in §3.3, we next discuss two fundamental parameters of an agent routing scheme, namely path correlation and path length.

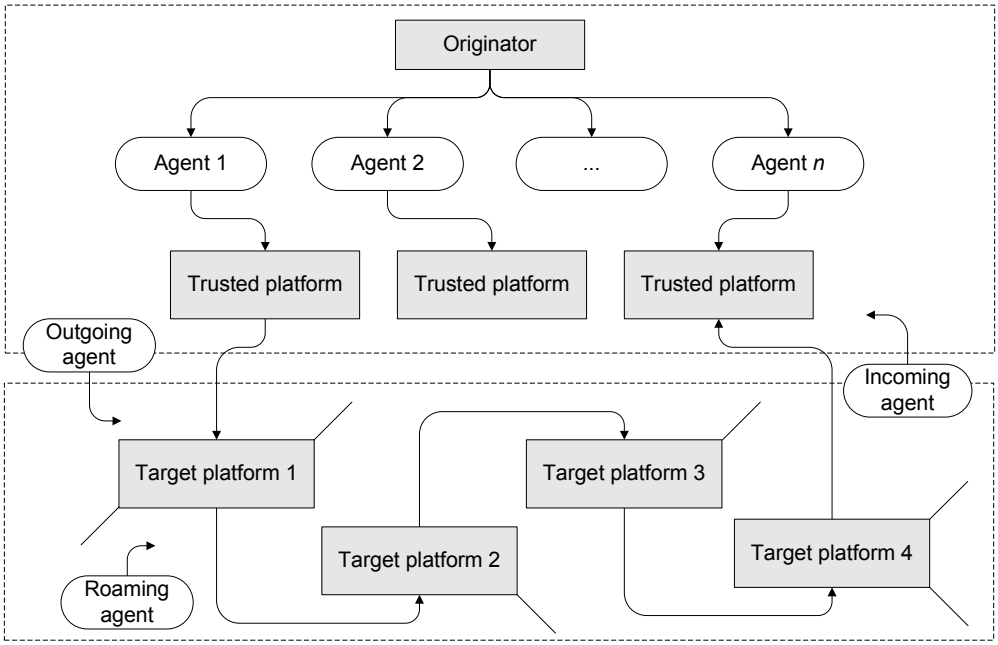


Figure 3.5: Spy agent routing architecture.

3.4.3.1 Path correlation

A migration path for an agent is a list of the target platforms that it is instructed to visit during its life (starting from a trusted platform). In order to minimise the probability of detection, peer spy agents visiting the same target platform should have minimal correlation, including between their migration paths.

Ideally, the information that target hosts retrieve from different visiting agents should be uncorrelated. To achieve this, it is necessary to know the nature of the inter-relationships between those target hosts that are to be visited by at least one of the spy agents. For example, we could assume that none of the target hosts will share information about visiting agents. In this case two spy agents would only need to be uncorrelated if these agents visit a common target host.

3.4.3.2 Path length

We assume throughout this thesis that spy agents will migrate through two or more target hosts rather than just one. We make this assumption for two main reasons. Firstly, we wish to try to minimise the number of spy agents required to assess a set of target hosts. Secondly, we assume that receiving an agent with only one destination will make a malicious host less likely to misbehave, since it can be held accountable for whatever happens to this agent. This is an important requirement that influences the design of agent routes.

Further, if a target host receives an agent that is programmed to migrate to another host not known to the target host (without migrating, for example, to a known competitor), then it will have a motive to refrain from behaving badly, either because it believes that this incoming agent might be a spy agent, or because it cannot identify any direct competition. Thus a malicious target platform might behave well to avoid possible detection, and the spy agent evaluation results will therefore be less likely to meet the system objectives. By contrast, a target host may feel free to misbehave if an incoming agent is programmed to migrate to a rival service provider. We also note that a trusted platform may not be able to be used in more than one host assessment process, lest a target host suspects that further incoming agents are spy agents, e.g. by making use of records of past events and/or statistical analyses.

3.4.4 Trust evaluation

The assessment of the trustworthiness of a target platform could yield estimates for a variety of security issues. Possible issues include the probability of the host reading or altering private data that should not be accessed, or the likelihood that it will block or divert migration of the agent.

These assessments could be made in a variety of ways, for example by comparing the data retrieved by a variety of agents using different routes. Also the returned agents could be examined to see if they have been altered in any way other than in terms of their retrieved data, such as blocking, delaying or changing a migration route. Alternatively, an agent could contain a temporary (unique) email address which is monitored to see whether spam emails are sent to this address in the future. If unsolicited emails are received at this address, then it can be deduced that one of the hosts visited by the agent containing this address may have violated the agent security policy by using private data in an unauthorised way. [The above ideas are developed further in Chapter 10.]

As discussed in the previous section, the information gathered from multiple agents visiting multiple hosts can be cross-referenced to make host assessments. The idea here is that if misbehaviour is detected involving one or more agents and the route design is selected with care, then the host platform(s) responsible can be identified with high probability. Alternatively, if an unmodified agent is returned as expected and there are no associated signs of mishandling, then it can be assumed with high probability that all visited target platforms have behaved properly.

The analysis of agent deployment can be either deterministic or probabilistic, and depends on the assumed spy agent scenario and the spy agent route design. Some fundamental spying scenarios are analysed in §5.3.

3.5 Conclusions

This chapter has introduced the concept of spy agents, and has described the possible benefits of their use. We have also described a generic agent structure

and discussed route design issues for spy agents.

A key issue for the proposed framework is the manner in which spy agents are coordinated to collect network information, from which conclusions about the trustworthiness of network entities can be drawn. It is proposed to use multiple spy agents coordinated by a number of trusted platforms, which will hide the identity and association of the agents.

In the future, spy agent networks can be used to offer security services to trusted networks, the price of which will depend on the required service credibility. Further, evaluation assessments of high quality (credibility) could be used by other applications in order to adapt their security to existing circumstances, and to perform further security assessments such as remote surveillance and risk analysis. Ultimately, it should be possible not only to evaluate a target host, but also to determine the benefit of using a specific quantity of resources within a specific spying scenario.

The remainder of this thesis expands on the introduced spy agent paradigm as follows. Chapter 4 provides a more detailed description of the fundamental spy agent system requirements. In Chapter 5 we introduce metrics which are designed to try to quantify the effectiveness of a specific spy agent route design. Techniques for designing efficient route designs for specific spy agent scenarios are given in Chapters 6, 7 and 8. Chapter 9 considers the credibility of spy agent results. Spy agent applications are discussed in Chapter 10, and overall conclusions are drawn in Chapter 11.

"Most of the research which is done is determined by the requirement that it shall, in a fairly obvious and predictable way, reinforce the approved or fashionable theories."

Celia Green

4

Spy agent requirements and assumptions

Contents

4.1 Synopsis

This chapter defines the fundamental spy agent security requirements, and gives our assumptions regarding the deployment scenario. Part of the work described in this chapter has been published in [100].

The spy agent security requirements are derived from the principles discussed in Chapter 3; these provide the basis upon which the spy agent routing problem is formulated in Chapter 6. First, the assumptions regarding malicious host behaviour are given in §4.2. The spying requirements are then classified into three groups: spy agent dissemblance requirements (see §4.3); host evaluation requirements (see §4.4); and spy agent routing requirements (see §4.5). Next, §4.6 discusses the security requirements for spy agent system operations that take place in trusted environments. The chapter concludes with a discussion of routing implementation issues and assumptions (§4.7).

4.2 Malicious host behaviour

Spy agents are intended to be used to determine whether or not a target host is malicious by observing the outcome of the host's behaviour towards visiting spy agents.

One key type of misbehaviour considered in this thesis is the malicious use of agent data (by hosts). This is defined as using agent information in an unintended manner, including the unauthorised transfer of information. Such misuse could include the unlawful use of, or access to, information, which might result in data protection, copyright, or privacy infringement. We note that in some cases it may be difficult to distinguish between deliberate (malign) behaviour and accidental misuse of data.

For our purposes, unauthorised handling of personal data will be regarded as malicious regardless of the intention of the host, on the basis that it is the duty of the authority holding sensitive information or code to protect it against unauthorised access or use (on behalf of the person or agent that the data or code relates to). This approach is consistent with laws which state that the data holder must take 'reasonable precautions' to protect sensitive data (see §2.5.1.2). For example, sensitive data stored on a server should be encrypted, and sufficiently strong authenticated access control mechanisms should be in place to ensure that stored data is not accessed by malicious parties.

In this thesis we thus assume, for simplicity, that hosts are responsible for any intended or unintended infringement of their data privacy policy and, hence, hosts allowing such infringements are either malicious or untrustworthy. For convenience, these terms are used interchangeably.

4.3 Spy agent dissemblance requirements

The spy agent dissemblance requirements analysed in the following subsections cover both the appearance and behaviour of a spy agent. These requirements are derived from the principles outlined in §3.3.

4.3.1 Subterfuge requirements

In order to be effective, spy agents need to be read and executed by remote hosts in the same way as any other agent. Spy agents should therefore provide remote hosts with no evidence of their purpose (see also §2.5.3).

One important subterfuge property is anonymity. Spy agents need to maintain their anonymity by hiding contextual information that could compromise their identity. This can be achieved using techniques such as the inclusion of fabricated context information.

Spy agents should operate in a non-malicious manner, as discussed in §3.3. Hence, a spy agent should use subterfuge only to protect its true identity and not to deceive a host into providing it with information that the host would not wish to reveal.

4.3.2 Statutory requirements

A spy agent must comply with the privacy and security requirements of a visited host, and conduct itself in accordance with stated policies. Software spy agents, unlike spyware, should not attempt to compromise the security of remote hosts, regardless of whether or not these remote hosts are malicious. A spy agent's task is the assessment of host behaviour without violating the host's policy or security protection mechanisms.

As part of these requirements, target hosts, including malicious hosts, may have the right to deny access to mobile agents if these agents do not conform to the host's public policy and requirements. For example, target hosts may block agents that exceed their computational limits in order to protect themselves against DoS attacks. Legitimate spy agents should not exceed such agreed limits.

A spy agent should only retrieve information from remote hosts in the manner offered by the hosts, while the hosts should not be given any evidence of the spy agent's true objectives.

The statutory spy agent requirements to some extent reinforce the subterfuge requirements. Any attempt to breach a host's security might cause the host to suspect that the mobile agent is not a 'normal' agent. In such a case the malicious remote host may adapt its behaviour to avoid detection, or simply to avoid an increased risk.

4.3.3 Protection requirements

Spy agents should comply with standard mobile code security requirements, including the incorporation of any standard mobile agent protection controls (see §2.4).

Protection requirements (indirectly) stem from the subterfuge requirements discussed in §4.3.1. If spy agents do not comply with standard mobile code security requirements, then it may be possible to distinguish them from other mobile agents. Also note that an abused agent may only be able to hold a malicious host accountable for its misbehaviour if the agent incorporates all reasonable protection precautions.

However, spy agents may not necessarily comply with all standard mobile code requirements in interactions with trusted networks and authorities; they only need to do so when interacting with target hosts. For example, spy agents might carry certified credentials bound to a fabricated ID.

4.3.4 Incentivisation requirements

Reasonable motivation should be given to malicious hosts to exhibit unauthorised behaviour. If a host is likely to violate the security of a normal mobile

agent, then it should be given an incentive to violate the security of a spy agent. Spy agents can use two techniques to achieve this: subterfuge (see §4.3.1) and incentivisation.

The nature of the required incentives will be application-dependent. For example, consider the following scenarios.

1. A malicious host with a good reputation might wish to frame an innocent rival host (especially if the innocent host already has a bad reputation).

2. A malicious host might wish to influence the result of a shopping agent so that the agent perceives that host more favourably than would otherwise be the case.

3. A malicious host might have a secondary unintended use for data contained within an agent.

4. A malicious host might wish to reduce its processing burden.

In the above examples, the incentive for misbehaviour can be increased in 1) by sending a spy agent to a large number of target hosts both reputable and non reputable; 2) by including within a spy agent high value inquiries which are potentially lucrative for the host; 3) by including within a spy agent information that could be useful to third parties (although the inclusion of large quantities of spurious data might arouse host suspicions); and 4) by sending processor-intensive agent code (which, however, should still induce a 'reasonable' computational expense, as discussed in §4.3.2).

Subterfuge requirements reinforce the incentivisation requirements. By encouraging hosts to believe that they can 'safely' misbehave, they are in some sense given an incentive to misbehave.

4.3.5 Summary of dissemblance requirements

The fundamental dissemblance requirement is that target hosts should be unable to exhibit special behaviour in order to make a false positive impression. Summarising the above analysis, we can expand the spying principles discussed in §3.3 to obtain the following list of spy agent requirements.

- Target hosts should be incapable of deciding whether or not they are dealing with spy agents.
- Potentially malicious target hosts should be given a motive to misbehave by using spy agents as 'bait'.
- Spy agents should appear to be 'normal' (e.g. m-commerce) mobile agents, and use full standard mobile code protection.
- Spy agents should use pseudonymous identities and credentials created in a legitimate manner.
- Spy agents should comply with all security requirements and behave as expected by the hosts.
- Information or feedback should be gathered and processed by the spy agent originator in a legitimate manner.
- Feedback should always be analysed in a safe environment.

These fundamental spy agent security requirements are analogous to those for honeypot systems, which "are nothing more than a security resource whose value lies in being probed, attacked, or compromised" (see §2.5.3.1), and those for decoy systems which use the notion of deception as the "deliberate misrepresentation of reality done to gain a competitive advantage" (see §2.5.3.2).

Spy agents differ in that attacks on mobile code take place in an adversary's environment rather than a safe one.

4.4 Spy agent evaluation requirements

The dissemblance security requirements discussed in §4.3 form one set of inputs to the design of a spy agent system. In this and the following section (§4.5) we consider other design constraints, namely evaluation and routing requirements.

4.4.1 Attack detection requirements

A malicious host can only be identified if the occurrence of an attack is detected. Hence spy agents are only useful in scenarios where a malicious act yields a detectable impact.

A taxonomy of possible attacks on mobile agents is given by Borselius [18] (see also §2.4.2.1). The impact of an attack depends on the scenario. For example, unauthorised manipulation of agent code [36] might be detectable, or the delayed return of an agent could imply deliberate retention of an agent for malicious processing [56]. However, it is not always possible to detect an attack. As discussed above, spy agents are only useful in scenarios where attacks result in detectable impacts, and the origin of such attacks (i.e. one or more malicious host(s)) can then be inferred by examining the results of all the spy agents.

Note that it is not necessary for all the agents to arrive back in a safe environment in order to complete the analysis. In an appropriate scenario, a lost agent could be considered as a positive outcome. Further, for the purposes of attack detection, spy agents could be combined with state appraisal or detection object techniques, as discussed in §2.4.4.2.3. Examples of such

combinations are further discussed in Chapter 10.

4.4.2 Attack identification requirements

In a spy agent system, the outcomes of spy agent tests are combined to deduce which hosts are (likely to be) misbehaving. The information obtained from an attack on a single spy agent is, in general, insufficient to assess visited hosts. Indeed, if an attack on a single agent could be linked with certainty to a particular host then that host would have no motive to misbehave, as discussed in §4.3.4. Hence, in order to meet both the dissemblance and attack identification requirements, more than one spy agent will be needed. The combined test outcomes should allow the origin of detected attacks to be identified.

4.4.3 Host evaluation fairness requirements

Regardless of whether or not certain hosts are believed to be malicious (e.g. by reputation or previous spy agent evaluations), we suppose that a new spy agent evaluation analysis will be based on the assumption that all target hosts are equally likely to be malicious. This 'fairness' requirement does not preclude further uses of evaluation results by other schemes, such as reputation systems.

4.4.4 Security evaluation optimisation requirements

Spy agents should be designed to yield evaluation results that are as correct as possible, given the resources used. An error in host trustworthiness evaluation could arise in various ways, including:

- an assessment error, e.g. occurring from the uncertainties of a probabilistic evaluation; and

- a model error, arising when a malicious host behaves in an uncharacteristic way, e.g. to avoid detection.

To alleviate the above problems, a spy agent system should be designed to minimise the probability of a) obtaining erroneous outcomes and b) making uncertain assessments, and to maximise tolerance to such errors. For example, if malicious target hosts misbehave inconsistently, spy agents should be designed to either detect this inconsistency (error detection) or, if possible, still make correct evaluations (error correction).

4.5 Spy agent routing requirements

As the outcomes of more than one agent need to be combined to identify the origins of attacks (as discussed in §4.4.2), spy agent route selection is a critical issue (see §3.4.2).

☞ For convenience, the terms 'routing' and 'migration' are used interchangeably. Also the term *spy agent routing* is used to refer to the set of migration paths assigned to the spy agents used in a single instance of the system.

Spy agents are assumed to have a predetermined migration logic, which will be available to all visited target hosts. A spy agent is expected to be executed on visited hosts as specified by its originator, unless one of these hosts maliciously changes the agent's code or migration logic. That is, a spy agent should only be accessed by the hosts it is expected to visit, unless one of the specified hosts maliciously modifies it. This agent routing security requirement can be guaranteed through the use of standard cryptographic

protocols that hosts should be required to use. Violation of this protocol is likely to be detectable (see, for example §2.4.3.3.6 and §2.4.3.3.5).

Additional routing requirements can be divided into the following two categories.

- Single-agent routing requirements: cover the security aspects of agent communication and migration.
- Multi-agent routing requirements: cover the migration logic for each agent in a single instance of the spy agent system.

4.5.1 Single-agent routing requirements

As discussed above, we assume that if a spy agent is misused then one of the hosts on its predetermined migration path must be responsible. Again as stated above, this assumption can be satisfied by requiring all hosts to use secure communications protocols to transfer agents.

In line with the spy agent protection requirements (§4.3.3), the migration logic of each spy agent (as provided by the originator) should be fixed. Given this, and as above, if the migration path is in any way modified then it is assumed that at least one visited host has misbehaved.

4.5.2 Multi-agent routing requirements

The collection of spy agents used in a single instance of the system will possess a set of routes designed to efficiently test the trustworthiness of a predefined group of target hosts. Hence, in theory at least, target hosts could pool information about visited agents to try to detect such a design and hence detect the use of spy agents. Thus sets of spy agent routes should be designed to minimise

the risk of a successful analysis of this type (as stated in §4.3.5). In particular, spy agent route sets should be designed so that peer spy agents visiting a target host have minimal common routing information (as also discussed in §3.4.3).

Algorithms for generating spy agent route designs form the core of the research described in this thesis.

4.6 Spy agent trust services

Spy agents should be trustworthy. That is, if spy agents do not comply with the statutory requirements (given in §4.3.2), then they should be classified as malicious agents. To guarantee spy agent trustworthiness, the following trust services should be provided.

4.6.1 Services in trusted networks

As discussed in §3.4.2, the active support of a number of trusted third parties may be required in order to set up a spy agent scenario. For example, if the spy agents require certified decoy credentials or bank accounts, then appropriate organisations will need to participate.

Although spy agents are anonymous (or pseudonymous), as discussed in §3.4.2, they should still be accountable for their actions. For example, violated target platforms should be able (with the help of trust services) to identify the source of malicious spy agents. In general, the following trust services should be provided.

- Spy agent accountability: spy agent originators, or any parties that obtain decoy IDs and certificates, need authenticate themselves to the authorities that provide them with these credentials. Spy agent originators

should not be able to repudiate their actions.

- Integrity of spy agent results: spy agents should not be able to modify spying outcomes.
- Confidentiality of spy agent results: spy agent outcomes and evaluation results should only be accessible to authorised (trusted) parties.

4.6.2 Evaluation services

Spy agents constitute a passive preemptive trust assessment mechanism that detects but does not thwart security attacks. However, the very existence of such evaluation mechanisms can in the long run contribute towards preventing malicious acts from taking place.

This prevention mechanism (like all schemes involving detection after the event) will still allow target hosts to misbehave. However, our assumption is that hosts are expected to eventually develop good behaviour regardless of whether or not they are actually assessed.

Evaluation results can be used for post-data processing and in the provision of other security services, such as *risk management* and *digital insurance* services. However, care should be taken to avoid biased evaluations (see §4.4.3).

4.7 Other assumptions

4.7.1 Agent anonymity and host identification

As discussed in §3.4.1, a spy agent needs to be anonymised before it is despatched to a target platform. How anonymity is provided depends on how the spy agent is identified by a target platform. For example, if the agent ID is supported by a public key certificate, the associated certificate chain can be used to help authenticate the agent's identity.

The means that can be used to anonymise an agent will depend on the nature of its ID. For example, if the ID is a URL (or URI) then the originator could use NAT [167], visualisation [196], dynamic DNS [178], and/or anonymity networks such as onion routing (see, for example, [142]) to provide anonymity. The trusted platforms (with individual URLs) through which the originator despatches its spy agents may not necessarily be mapped to individual physical hosts; in this case a platform ID refers to a logical rather than a topological entity. Multiple hardware devices could share the same logical identity within a subnet, and a single hardware platform could have multiple logical identities belonging to different subnets. A logical platform could physically exist anywhere within a network, and a network could consist of many logical or physical platforms.

As a result, in general, trusted platforms and spy agents will not be able to verify the real physical identity of a platform, as this may be encapsulated within a complex network. Hence, spy agent evaluation results will apply to virtual hosts; in such a case the detection of a physical malicious entity may require further network identification procedures, such as ISP enquiries. The scope of such further procedures is outside the scope of this thesis.

It should be noted that, in order to avoid target hosts sharing information, the identifiers of two logical target hosts should preferably correspond to different physical hosts. If information is shared between hosts, then the difficulty of correctly evaluating these hosts is increased. [The problem of host collusion is discussed further in Chapter 7, where a possible solution is given.] Since it is common practice to have multiple physical hosts and multiple local identities for redundancy and resilience to single point failure, a spy agent originator should ideally be aware of this network configuration information.

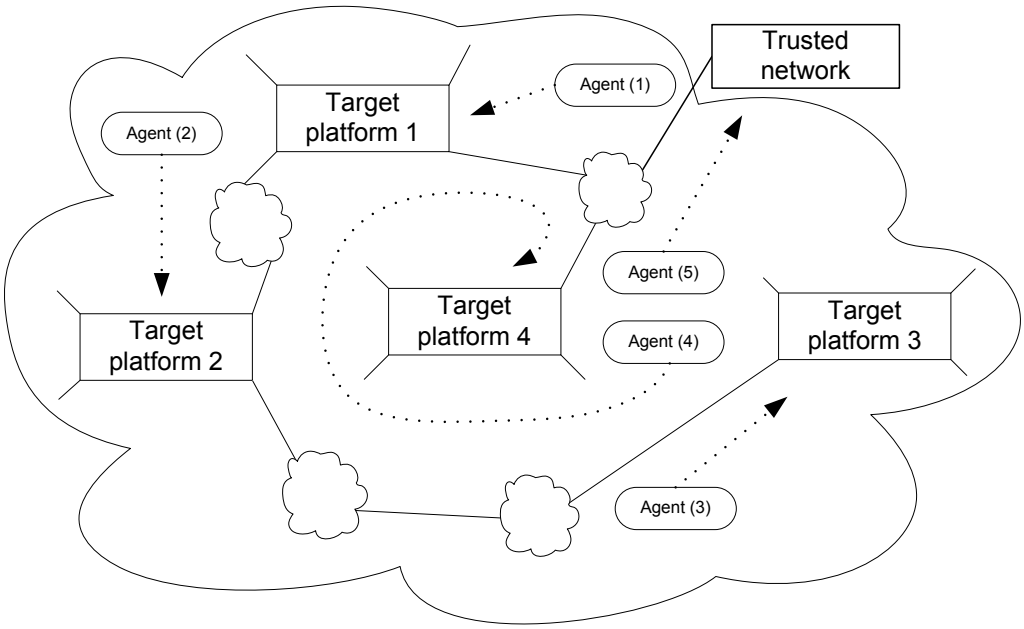


Figure 4.1: Spy agent virtual route.

In such a case such linked identifiers can be treated as a single logical host. Finally, spy agent models should take into account the possibility of other errors in assumptions made about the network and the target hosts (see §4.4.4).

☞ Henceforth, terms such as nodes, platforms, and hosts (either target or trusted) will be used to refer to (logical) entities that have distinct public IDs, and which can be authenticated using public key certificates or other cryptographic credentials.

4.7.2 Inter-networking assumptions

A spy agent route (as shown in Figure 3.5) may require the agent to be routed through a complex network before reaching the target platforms. Such a scenario is depicted in Figure 4.1. Provided that secure end-to-end agent communication protocols are used, as discussed in §4.5.1, it seems reasonable to assume that intermediate routing entities do not affect agent migration.

4.8 Conclusions

This chapter has set out the spy agent system requirements and assumptions regarding the usage scenarios. These requirements are used in the next chapter to analyse certain fundamental spy agent protocol architectures, and are used in Chapter 6 to motivate the design of a spy agent routing model.

"The value of a principle is the number of things it will explain."
Ralph Waldo Emerson

5 Protocol architectures and analysis principles

Contents

5.1 Synopsis

This chapter analyses certain fundamental spy agent protocol architectures, using the spy agent framework given in Chapter 3 and the spy agent system requirements given in Chapter 4. Aspects of the work described in this chapter have been published in [101].

The rest of the chapter is organised as follows. Section 5.2 outlines implementation principles; §5.3 analyses a number of spy agent protocol architecture implementation scenarios; and §5.4 describes certain architectural parameters and metrics that can be used to quantify the effectiveness of a specific routing protocol.

5.2 Scenario implementation issues

As discussed in §4.3.4, a spy agent should be designed to encourage malicious hosts to misbehave, in order to obtain the most accurate assessment of their genuine behaviour. Furthermore, the very existence of spy agents seems likely to mitigate the existence of malicious service providers, since they will be unable to distinguish between spy agents and normal e-commerce agents. Hence it will not be possible for them to know if they can breach security policies without being detected.

The design requirement that target hosts should be prevented from determining whether or not they are dealing with a spy agent, may run counter to the requirement to make precise trustworthiness assessments of the target platforms. A deterministic assessment could be made by sending a number

of spy agents to just one target host; however, such a strategy is likely to raise suspicion in the target host, potentially resulting in atypical behaviour. On the other hand, it seems reasonable to assume that when processing spy agents that migrate via a large number of competitor hosts, malicious hosts are more likely to exhibit characteristic behaviour. However, in the latter case, assessments of individual hosts become significantly more difficult to achieve.

Clearly an optimal protocol architecture should try to balance the various conflicting requirements to obtain the best strategy. The optimal strategy will almost certainly vary depending on a range of factors, including: how many trusted devices a mobile terminal has, what the computational costs are, what the objective of the analysis is (e.g. to maintain high quality security profiles or to perform an ephemeral test), when the results are needed (e.g. immediately or within a fixed time period), the required accuracy of the results, and how much the spy agent originator is willing to pay for the results.

5.3 Fundamental spy agent protocol architectures

5.3.1 Single-agent-single-target scenario

In this scenario (shown in Figure 5.1) a single agent migrates to a single target host before returning to a trusted host. The single-agent-single-target host scenario is not interesting for our purposes, since it will be clear to the target host that it can be held accountable for any detectable malicious act involving the agent or its data.

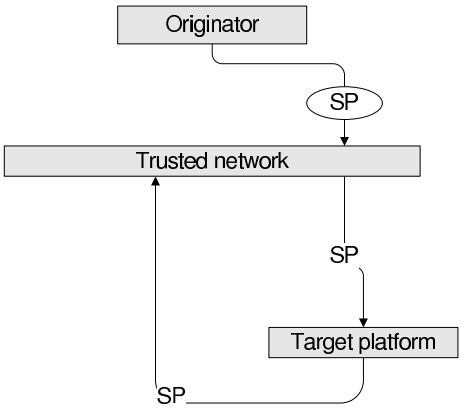


Figure 5.1: Testing a target platform with the aid of a trusted platform.

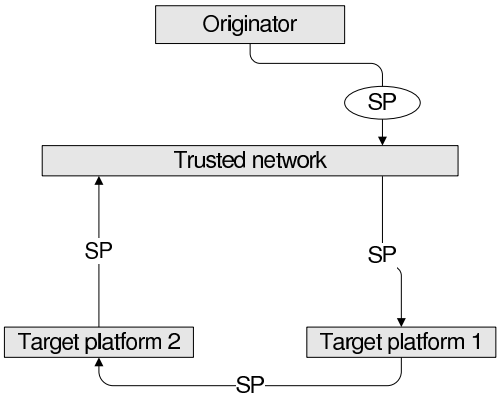


Figure 5.2: Testing two target hosts with a single agent.

5.3.2 Single-agent-two-target scenario

If a single spy agent migrates to two (or more) target hosts (as shown in Figure 5.2), a malicious host might reasonably decide that it can risk misusing the agent since it will only be jointly accountable for any misbehaviour with the other host(s); indeed, it may even be able to 'frame' another host. However, this scenario does not meet the attack identification requirements (§4.4.2), since the agent originator will be not be able to uniquely identify a misbehaving host.

5.3.3 Unbalanced routing scenarios

An 'unbalanced routing scenario' is one in which spy agents are designed to evaluate (deterministically or stochastically) only a subset of the target hosts. This might result in one host being visited by more spy agents than another; in such a case the former host might be deemed to be unfairly targeted by spy agents.

Unbalanced routing scenarios do not meet the host evaluation fairness requirements, as discussed in §4.4.3.

5.3.4 Two-agent-two-target scenario

Results regarding multiple hosts can be obtained by deploying multiple agents. If two agents each visit two hosts, a minimum of two and a maximum of four targets can be tested, as discussed below.

1. The two agents could visit the same two platforms. If the order of visit is of no significance (as discussed in §5.4.4) or if the agents visit the hosts in the same order, then this scenario is equivalent to two identical instances of the single-agent-two-target scenario, as discussed in §5.3.2.

2. The two agents could visit four platforms. This scenario is equivalent to two independent instances of the single-agent-two-host scenario, as discussed in §5.3.2.

3. The two agents could visit three platforms. One platform is visited by two agents, whereas the other two platforms are only visited by one agent. This yields an unbalanced routing scenario, as discussed in §5.3.3.

It is interesting to further analyse case 1), if we assume that the two agents visit the same two platforms in reverse order. This is shown in Figure 5.3. The

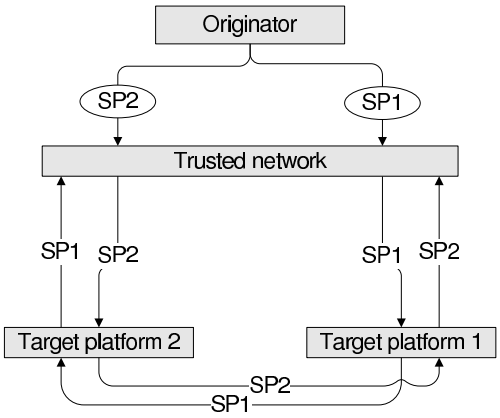


Figure 5.3: Testing two target platforms with two agents.

following subcases can be distinguished.

a. Suppose that the outcome of the spy agent $SP1$ is positive and the outcome of $SP2$ is negative. If the two target hosts do not behave consistently, then the two target platforms have an equal probability of being malicious. If, however, the two hosts behave consistently, it follows that the order of visit influences at least one host's behaviour (assuming there is no other significant difference between $SP1$ and $SP2$). The degree to which the order of visit influences the behavioural model depends i) on each target host's motivation to misbehave, and ii) on each malicious host's level of sophistication. If, for example, we assume that the second visited target platform has a motive to modify the previously visited host's input, and the first visited platform does not attempt to 'frame' the second visited platform, then we can infer that the second visited platform is more likely to misbehave. Hence, in this case target platform 2 is more likely to be malicious. More general versions of this example have been studied in the context of agent replication (see §2.4.3.3.1).

b. Suppose that both agents yield positive outcomes. This might mean

that both platforms have misbehaved. However, even if both hosts are potentially malicious, it might not be the case that both hosts have actually misbehaved if one target platform is sophisticated enough to modify its behaviour to disrupt the evaluation analysis.

c. Suppose that there are no signs of an attack in both agent outcomes. This suggests that both target hosts are well-behaved. However, this may not be the case if, for example, a potentially malicious target does not misuse an agent if the agent is due to visit just one other host, and this host has a good reputation.

5.3.5 Three-agent scenarios

While the single-agent-two-target scenario (see §5.3.2) is the simplest scenario that could in principle satisfy the spy agent incentivisation requirements (see §4.3.4) and the routing requirements (see §4.5), this scenario still does not satisfy the identification requirements (see §4.4.2). This is still the case if an additional spy agent is introduced (see §5.3.4). However, the situation changes with the introduction of a third target platform; this defines what we refer to as the *three-agent-three-target scenario*. This is the simplest spy agent routing scenario which can satisfy both the security and identification requirements for spy agents.

The three-agent-three-target scenario is shown in Figure 5.4. The originator creates and disseminates three spy agents via trusted platforms within the trusted network. Each agent migrates to two target platforms before returning to a host within the trusted network.

We denote the three target platforms by s_1, s_2 and s_3. We suppose that the first spy agent, $SP1$ say, migrates to the first and the third target platforms,

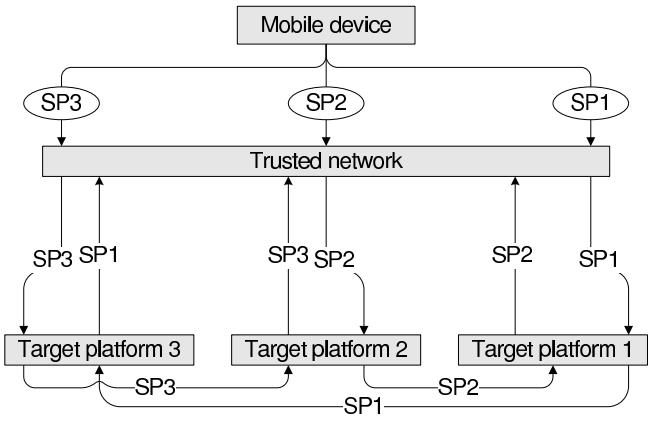


Figure 5.4: Three-agent-three-target scenario.

s_1 and s_3, before returning to the trusted network; $SP2$ migrates to s_2 and s_1; and $SP3$ migrates to s_3 and s_2.

Target platform s_1 is visited by $SP1$ and $SP2$, and is thus able to retrieve the following pair of migration paths: $(s_1, s_3), (s_2, s_1)$. Assuming that target hosts do not collude, s_1 cannot observe any correlation between the routing behaviour of $SP1$ and $SP2$ since it is the only common element in the two routes. As a result of the symmetry of this route design, similar assertions can be made for s_2 and s_3.

Assuming that the target platforms demonstrate characteristic behaviour and do not modify their behaviour to avoid detection (e.g. as might be the case if they suspect the purpose of one or more of the spy agents), then this routing scenario enables us to make the following deductions. Suppose that $SP1$ and $SP2$ yield a positive result, but $SP3$ does not. Then, since $SP1$ visited s_1 and s_3 and $SP2$ visited s_2 and s_1, it can be deduced that s_1 is more likely to have misbehaved. In a scenario involving greater numbers of agents and hosts, it is envisaged that more detailed and fine-grained conclusions can be drawn by examining all the spy agents after they have completed migration, as long as it is possible to predict to what degree malicious target platforms

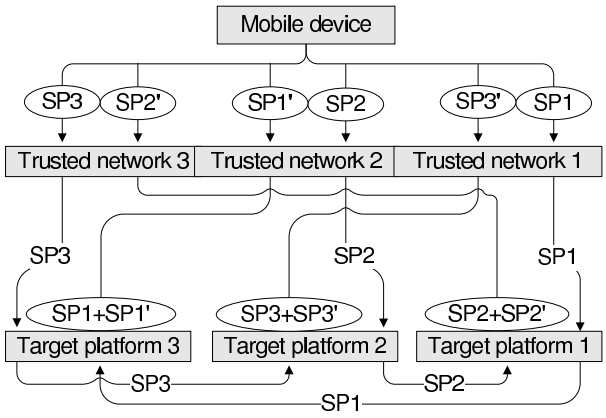


Figure 5.5: Three-agent-three-target scenario with guidance.

might modify their behaviour to avoid detection.

5.3.6 Guidance spy agent scenarios

The three-agent-three-target scenario (as discussed in §5.3.5) can be extended by introducing the notion of 'guidance spy agents'. In this scenario the originator uses three trusted platforms to support a distributed routing strategy, as well as to maintain anonymity. The originator instantiates six spy agents, separated into two matching groups of three (as shown in Figure 5.5):

- The first three spy agents start their migration from a single trusted platform and then each migrate to two target platforms in turn. The spy agents do not contain any logic dictating where they should migrate after visiting the second target platform.
- The three spy agents in the second group, called guidance spy agents, are in one-to-one correspondence with the spies in the first group, and contain migration information for the corresponding agents of the first group. They each visit the platform where their corresponding spy agent is waiting to be instructed where to go next. This technique enables the anonymity of the spy agents to be further enhanced.

This scenario is built on the assumption that two agents executing in a remote host can inter-communicate. It is also assumed that such communications are only allowed between two mobile agents that have been instructed in advance to do so. Hence, agent inter-communication will only be allowed between a spy agent and its respective guidance spy agent. Mobile agents could employ cryptographic techniques to thwart any breach of this requirement. For example standard cryptographic protocols can be used for mutual authentication and integrity (see for example, Boyd and Mathuria [20]).

Let s_1, s_2, s_3 and t_1, t_2, t_3 denote the target platforms and trusted platforms, respectively. The first spy agent, $SP1$, (see Figure 5.5) leaves the first trusted platform, t_1, visits the first target platform, s_1, migrates to s_3, and waits for the arrival of the first guidance spy agent, $SP1'$, which is sent from t_2. Analogously, $SP2$ starts from t_2 and migrates to s_2, then s_1, and waits for further instructions from $SP2'$, which is sent from t_3. In a similar way, $SP3$ migrates via s_3 to s_2 and waits for guidance from $SP3'$. Finally, $SP1$, $SP2$ and $SP3$ are instructed by their guidance agents to return to t_2, t_3 and t_1 respectively.

Target platform s_1 is visited by three agents: $SP1$, $SP2$ and $SP2'$. Hence, s_1 is able to retrieve the following information regarding the migration paths for $SP1$, $SP2$ and $SP2'$.

- $SP1 : (t_1, s_1, s_3)$
- $SP2 : (t_2, s_2, s_1, t_3)$
- $SP2' : (t_3, s_1, t_3)$

As in the three-agent-three-target scenario, s_1 cannot observe any correlation between the routing behaviour of $SP1$ and $SP2$, since s_1 is the only

common element in the two routes. It can observe that the routes of $SP2$ and $SP2'$ share t_3, but this does not look suspicious since $SP2$ is expecting guidance from $SP2'$. A similar argument applies to s_2 and s_3. Hence this routing design satisfies the basic spying requirements.

An obvious potential weakness of this routing design is the fact that the target platforms could learn about the spying scenario if they exchange information. However, this weakness can be avoided by employing more trusted platforms and designing the routes for each agent in such a way that the risk of platform collusion revealing spy agent behaviour is minimised.

5.3.7 Multiple target agent scenarios

As the number of spy agents and target hosts increases, the analysis of the results becomes increasingly difficult. This gives rise to the need for a methodology for designing sets of agent routes and for corresponding evaluation algorithms. A formal approach to this problem is presented in §6.3, and solutions are given in the chapters that follow.

We end this section by discussing a scenario in which two agents each visit three target hosts. In such a case a minimum of three and a maximum of six targets can be tested, as follows.

- The two three-target agents could visit the same three platforms. If the two agents visit the three hosts in a different order (i.e. there is no agent mirroring) and the order of migration has an effect on the outcome, then this is an unbalanced routing scenario, as agents do not visit hosts in all possible orders.
- The case where two agents visit six platforms is equivalent to two uncorrelated three-target agent scenarios.

- All remaining cases give rise to unbalanced routing scenarios.

5.4 Spy agent system parameters

The degree to which the spy agent system requirements discussed in Chapter 4 are met depends on a range of variables. In this section we analyse the parameters that determine critical properties of spy agent systems.

5.4.1 Number of spy agents

If used rationally, the larger the number of spy agents, the greater the amount of evidence about host behaviour is generated. On the other hand, a large number of spy agents with common elements visiting common target hosts may trigger suspicion amongst these hosts, and may therefore compromise the accuracy or credibility of the observed spy agent outcomes (see §3.4.3.1).

Thus selecting the number of spy agents is a trade off between maximising the generation of information and minimising possible host suspicion. Ideally, sufficiently many agents should be deployed to recover the desired evaluation information whilst minimising the exposure of the hosts to spy agent information.

5.4.2 Number of trusted platforms

As discussed above, increasing the number of trusted platforms used by a spy agent system helps to increase spy agent anonymity. However, there is a potential cost associated with every additional trusted platform used, and thus it is desirable if trusted networks can be re-used by spy agents, in a way that minimises their exposure to target hosts. In practice, however, it may be easier to assign one trusted network to every spy agent.

5.4.3 Number of target platforms

Increasing the number of hosts a spy agent visits may provide a greater incentive for a malicious host to misbehave, as discussed in §3.4.3.2. However, this might also make the evaluation analysis harder. Thus a long migration path is likely to improve the accuracy of evaluation at the cost of increasing the level of uncertainty about individual host behaviour. Ideally we need to achieve a route design which has adequately long migration paths and which can at the same time yield useful evaluation results.

5.4.4 Order of target platforms

The order in which spy agents visit target platforms may or may not be significant, depending on the nature of the evaluation. Examples in which the order might influence a spy agent assessment were discussed in §5.3.4 and §5.3.7. One application where the order of host visits may be significant is e-ticketing (see §2.4.3.3.1.2).

5.4.5 Cost and overheads

Cost is an essential factor in any kind of security evaluation. The cost of implementing a security mechanism should not be greater than the cost of the potential damage prevented by this mechanism. The cost of deployment of a spy agent scenario depends on the amount of resources used. Apart from the network and computational resources, spy agents require an infrastructure offering a number of trust services (see §4.5). The cost of such an infrastructure could be significant, given that the agent credentials and IDs should only be used once.

5.5 Conclusions

This chapter has introduced a number of fundamental spy agent routing architectures for the spy agent framework introduced in the previous chapter, and has analysed issues arising in their use. This analysis forms the basis for the adversarial modelling, and the spy agent routing problem formulation and evaluation given in the next chapter.

"Information is not knowledge."

Albert Einstein

6

Spy agent routing designs using non-adaptive group testing

Contents

6.1 Synopsis

This chapter introduces a methodology for constructing route designs for spy agents. The need for this methodology was discussed in §5.3.7.

In the design problem, a network of remote agent platforms are tested by roaming spy agents in order to identify those that are malicious, where the results are derived from observations of the outcome of each agent. As we discuss below, given a set of spy agent requirements (see Chapter 4), the task of choosing the sets of platforms which each spy agent visits can be abstracted as a GT problem (see §2.6). In particular, the applicability of NGT (see §2.6.3) is considered in detail, and a simple combinatorial construction for a set of agent routes is presented which combines known results from the prior art.

The rest of the chapter is organised as follows. Section 6.2 introduces the routing problem, §6.3 gives a mathematical formulation, and §6.4 presents a simple scalable NGT construction for sets of spy agent routes.

Most of the work described in this chapter has been published in [100].

6.2 Introduction to the routing problem

As discussed in §1.3, the main problem with remote security assessments (using roaming agents) is that a corrupted remote host could detect incoming security agents and selectively behave well in order to escape detection. The same host could behave inappropriately when it has the opportunity to cheat without being detected. Careful design of route sets for spy agents can help to address this issue, and we consider here the use of combinatorial GT methods (introduced in §2.6) to develop good route sets.

That is, the main focus of this chapter is the choice of sets of routes for spy agents which efficiently identify which remote hosts misbehave. For the purposes of the methodology presented in this chapter, a route is defined as an (unordered) set of hosts visited by an agent. This contrasts with the more usual definition of route, in which the order of destinations is of significance (see §5.4.4).

As discussed in previous chapters (and, in particular, in §5.3.1), a simple solution to this routing problem would be to send each spy agent to a single target host. The evaluation process for such a design is trivial, since a positive result for a spy agent implies a misbehaving host. However, improved solutions are sought for the following reasons:

- *Security*: Larger test groups yield more credible results. As discussed in §3.4.3.2, the longer the migrating route, the less likely it is that a malign host will suspect a spying scenario, and the greater the chance that it can cheat without being detected. As a result, it is more likely to misbehave, revealing its true character.
- *Economy*: Less agents and less time are needed if hosts are tested in groups instead of one by one. This is of particular importance when routinely performing large scale tests.

6.3 Problem formulation

6.3.1 Assumptions and objectives

We make the following assumptions when designing sets of agent routes.

- The order in which a spy agent visits a subset of target platforms is of no significance. Thus, the subset of target hosts to be visited defines a

route.

- A target platform is aware of the routes of all the spy agents that visit it.
- Potentially malicious target hosts do not share any information about visiting agents, and will not collaborate to attempt to avoid detection.
- A compromised spy agent will always provide information, or perform an act, which will enable the sender of the agent to detect that it has visited a malicious platform. That is, a spy agent visiting a malicious host will always yield a positive outcome. (This is a rather strong assumption, which we relax in Chapter 7.)

These assumptions enable us to consider spy agent routes as unordered sets of hosts that spy agents visit, where each set returns 'positive' when it contains at least one malicious host, and 'negative' otherwise. Clearly the problem of efficiently identifying the malicious (defective) hosts is identical to the classic GT problem (see §2.6.1).

Optimal sets of spy agent routes can be designed using 'efficiency criteria', 'security criteria' or a combination of both. In terms of efficiency, the following criteria are adapted from the GT literature (see, for example, [128]):

(Eff-1) A design is deemed optimal if it can be used to test the maximum possible number of target hosts using a given number of spy agents.

(Eff-2) Alternatively, we can define a design to be optimal if it can reliably detect the maximum number of malicious hosts from amongst a given number of target hosts.

From the spy agent dissemblance requirements listed in §4.3.5, and the routing principles for path correlation (see §3.4.3.1) and path length (see §3.4.3.2), the following security criteria can be derived.

(Sec-1) The number of hosts that each spy agent visits should be maximised. The rationale for this is that the more target hosts that a spy agent visits, the greater is the likelihood that the target host is unable to distinguish it from a 'regular' agent and, hence, the more reliable the tests are.

(Sec-2) The number of hosts in common between any pair of spy agent routes should be minimised. This follows from the assumption that target hosts will be less likely to be able to distinguish spy agents from 'regular' agents if spy agents appear to be as different from one another as possible. Ideally, any two spy agents should visit no more than one common target host.

From the above it is clear that the spy agent route optimisation problem is a version of the classic GT problem, modified by the inclusion of some additional security optimisation criteria.

6.3.2 Group testing scenario hypotheses

As described in §2.6, there are many different GT algorithms, including sequential and non-adaptive schemes; the choice of algorithm depends on the application. In this chapter the following assumptions regarding the usage scenario are made.

- Completing a single test may take a long time.

- A test outcome that is initially negative may, after some unidentified time, change to positive. In such cases, adaptive tests may perform poorly from both an efficiency and a security point of view.
- The assumption that malicious hosts always process agents in a detectably malicious way is more likely to be valid within a longer time frame.

These hypotheses may characterise, for example, a scenario where each spy agent contains a unique decoy email address and the test for target hosts is whether there is a violation of PII privacy (see §2.5.1); this application is discussed further in Chapter 10. In this case, the impact of the misuse of an email address could be realised at any time after the spy agent carrying this address has commenced its migration.

It is clear that in scenarios such as the one described above, NGT designs are preferable. There is a rich literature on such schemes and their applications [46] (see also §2.6.5.4). Constructing optimal examples of the various classes of design is an ongoing research topic, and comparing different schemes is a non-trivial subject [128].

It is important to note that NGT designs are usually deemed to be 'good' if they involve a relatively small number of tests. Such designs may also possess relatively long routes, in which case they are, in addition, 'good' spy agent route designs (by the above criteria). For example, Balding et al. [8] have shown that maximum sized *packings* correspond to NGT designs containing a very small number of tests.

In this chapter we propose the use of a rather simple NGT construction based on well-known combinatorial block designs. We first give some fundamental notation and describe certain properties of such designs.

6.3.3 Definitions and fundamental results

6.3.3.1 Route design and incidence matrix

We define a special class of design to represent the incidence of hosts (items) with network routes (tests) as follows. Let S be the set of n items (hosts), i.e. $|S| = n$. We define a spy agent *route design*, $\mathcal{H}$, to be a set system $\mathcal{H} = (S, \mathcal{R})$, where $\mathcal{R}$ (the set of routes) is a set of subsets of S. The route design is said to be *uniform* if all routes contain the same number of hosts, i.e. $|R_i| = |R_j|$ for all $R_i, R_j \in \mathcal{R}$. The cardinality of $\mathcal{R}$, $|\mathcal{R}|$, is simply the total number of spy agents (i.e. one spy agent is assigned to each route).

A route design may be represented by an *incidence matrix* $M = (m_{ij})$, whose rows are labelled by routes (tests) and columns by hosts (items), and where $m_{ij} = 1$ if route R_i contains host j, and $m_{ij} = 0$ otherwise. The ith row is the incidence vector, $\boldsymbol{r}_i$, of the subset $\{j|m_{ij} = 1\}$ (representing route R_i), and the jth column is the incidence vector, $\boldsymbol{c}_j$, of the subset $\{i|m_{ij} = 1\}$ (representing the jth host).

6.3.3.2 Combined set and binary vector operations

Standard set theory notation can be obtained from Jech [90]. Also, elementary binary vector notation is given in §2.6.5.2. We extend this notation by combining set and binary vector notation in the following way.

We identify row and column vectors ($\boldsymbol{r}_i$ and $\boldsymbol{c}_j$) with the routes (R_i) and hosts, so that we refer to a route $\boldsymbol{r}_i$ and a host $\boldsymbol{c}_j$. We also abuse our notation further and apply set operations to these binary vectors. For example, the union (intersection) of two (or more) incidence vectors corresponds to applying the bit-wise boolean OR (AND) operation to these vectors. In the same way, we say that a column $\boldsymbol{c}_i$ *contains* another column $\boldsymbol{c}_s$ if the column $\boldsymbol{c}_s$ contains

no row that is not contained in $\boldsymbol{c}_i$, or, equivalently, if $\boldsymbol{c}_i \cup \boldsymbol{c}_s = \boldsymbol{c}_i$. For any vector $\boldsymbol{v}$ we use $\boldsymbol{v}[i]$ to refer to the ith element of $\boldsymbol{v}$. For example, $\boldsymbol{r}_i[j] = \boldsymbol{c}_j[i] = m_{ij}$. Finally, the size of an incidence vector refers to its Hamming weight, i.e. the cardinality of the corresponding subset, i.e. $|\boldsymbol{r}_i| = |R_i|$.

6.3.3.3 Representing defective items

Given a set S of n hosts, the set of defective items can be represented as a binary vector $\boldsymbol{\delta} = (\delta_1, \delta_2, \ldots, \delta_n)$, where $\delta_j = 1$ if host $\mathbf{c}_j$ is defective, and $\delta_j = 0$ otherwise. The outcome of all tests can also be represented by a vector $\mathbf{a} = (a_1, a_2, \ldots, a_{|\mathcal{R}|})$, known in this case as the *outcome vector*, where $a_i = 1$ if route $\mathbf{r}_i$ contains a defective host and $a_i = 0$ otherwise, i.e. $a_i = \{i : |\mathbf{r}_i \cap \boldsymbol{\delta}| > 0\}$. The union of all columns corresponding to defective hosts is thus equal to the outcome vector. This follows from the fourth assumption given in §6.3.1, i.e. that if a host is defective then all routes containing it will yield a positive outcome.

Example 6.1. Consider the three-agent scenario discussed in §5.3.5. In this case $S = \{s_1, s_2, s_3\}$ and the set of spy agent routes consists of all 2-subsets of hosts. There are thus three spy agent routes: $\boldsymbol{r}_1 = (1, 1, 0)$, $\boldsymbol{r}_2 = (1, 0, 1)$ and $\boldsymbol{r}_3 = (0, 1, 1)$. The incidence matrix is given in Table 6.1. Suppose that s_3 is the only defective host, i.e. $\boldsymbol{\delta} = (0, 0, 1)$. Clearly, $\boldsymbol{\delta}$ is only contained in $\boldsymbol{r}_2$ and $\boldsymbol{r}_3$. Hence, the outcome vector is $\mathbf{a} = (0, 1, 1)$. In this simple example, given the outcome vector of the route design it is trivial to identify the defective host. However, this route design cannot identify the defectives if there is more that one defective host. In such a case we will always have $\mathbf{a} = (1, 1, 1)$ and we cannot determine whether $\{s_1, s_2\}$, $\{s_1, s_3\}$, $\{s_2, s_3\}$ or $\{s_1, s_2, s_3\}$ is the set of defectives.

Table 6.1: Simple three-agent scenario incidence matrix.

	$\boldsymbol{c}_1$	$\boldsymbol{c}_2$	$\boldsymbol{c}_3$
$\boldsymbol{r}_1$	1	1	0
$\boldsymbol{r}_2$	1	0	1
$\boldsymbol{r}_3$	0	1	1

6.3.3.4 Classifier route designs

We define two classes of spy agent route designs which are useful for host evaluation.

Definition 6.2. *Suppose that there are at most d malicious nodes in a network of n nodes ($n \geq d$). If the outcome vector, $\mathbf{a}$, of the incidence matrix, M, of a route design, $\mathcal{H}$, can be used to successfully identify all the malicious and honest nodes in the network, regardless of how they are distributed, then $\mathcal{H}$ is said to be a* $\overline{d}$-classifier.

Definition 6.3. *Suppose that there are exactly d malicious nodes in a network of n nodes ($n \geq d$). If the outcome vector, $\mathbf{a}$, of the incidence matrix, M, of a route design, $\mathcal{H}$, can be used to successfully identify all the malicious and honest nodes in the network, regardless of how they are distributed, then $\mathcal{H}$ is said to be a* d-classifier.

Lemma 6.4. *If route design $\mathcal{H}$ is a $\overline{d}$-classifier then $\mathcal{H}$ is a d'-classifier for all $1 \leq d' \leq d$.*

Proof. This follows immediately from Definitions 6.2 and 6.3. □

In group testing terminology, a $\overline{d}$-classifier is a $(\overline{d}, n)$ NGT design, and a d-classifier is a (d, n) NGT design (see §2.6.1 or [46]).

Theorem 6.5. (Hwang et al. [94]) *Let S be a set of n items containing a set D of defectives. We refer to the (d, n) group testing problem if D can*

be any d-subset of S, and the $(\overline{d}, n)$ problem if D can be any k-subset of S, $0 \leq k \leq d$. Let q be a group testing procedure that identifies D, and let M_q be the maximum number of tests in q. Then, for each procedure q for the (d, n) problem, there exists a procedure q' for the $(\overline{d}, n)$ problem such that

$$M_q + 1 \geq M_{q'} \,. \tag{6.3.1}$$

From the above theorem we deduce that if a design is a d-classifier, then there exists another design that includes at most one further test and is a $\overline{d}$-classifier. On this basis, in some cases below we restrict our attention to d-classifiers or $\overline{d}$-classifiers.

The problem of designing an optimal spy agent route design (as defined in §6.3.1) can be summarised as follows:

Problem 6.6. *For any given d and n, find a $\overline{d}$-classifier route design $\mathcal{H}$ satisfying one or more of the following criteria (where the choice of criteria depends on the scenario):*

1. *For given other parameters, maximize $\min_i |R_i|$ (derived from (Sec-1)).*
2. *$|R_i \cap R_j| \leq 1$ for every pair of distinct routes $R_i, R_j \in \mathcal{R}$ (derived from (Sec-2)).*
3. *For given other parameters, maximize n (derived from (Eff-1)).*
4. *For given other parameters, maximize d (derived from (Eff-2)).*

6.4 A spying classifier design

6.4.1 Properties of classifiers

In order to show how to construct a $\overline{d}$-classifier or a d-classifier, we first give some key properties of the incidence matrix, M, of a route design, $\mathcal{H}$. The

results in this section are adapted from §2.6.5.3, and are taken from Kautz and Singleton [105] (who use a somewhat different terminology).

Definition 6.7. *An incidence matrix M is said to be d-separable if the unions of the subsets of exactly d columns are all distinct.*

Definition 6.8. *An incidence matrix M is said to be $\overline{d}$-separable if the unions of the subsets of at most d columns are all distinct.*

Lemma 6.9. *If the incidence matrix M is $\overline{d}$-separable then M is d'-separable for all $1 \leq d' \leq d$.*

Theorem 6.10. *A route design H is a d-classifier if and only if its incidence matrix M is d-separable.*

Theorem 6.11. *A route design H is a $\overline{d}$-classifier if and only if its incidence matrix M is $\overline{d}$-separable.*

Example 6.12. Following from the analysis in Example 6.1, the 3×3 incidence matrix given in Table 6.1 is 1-separable and a 1-classifier, but is neither 2-separable nor a 2-classifier. Also, this incidence matrix is an optimal 1-classifier for the $S(1,3)$ problem, i.e. the problem of identifying at most one defective amongst a set of three hosts, in the sense that it maximises the route length.

One problem with the use of $\overline{d}$-separable designs (i.e. route designs with a $\overline{d}$-separable incidence matrix) is that no efficient general *decoding algorithm* is known for such designs, where a decoding algorithm takes as input an outcome vector and gives as output the unique set of malicious hosts (of size at most d) which could have given rise to this outcome. The trivial decoding algorithm (involving considering every possible subset of defective hosts) has complexity exponential in n. As a result we consider a slightly more restrictive class of designs for which we have a simple and efficient decoding algorithm.

Definition 6.13. *An incidence matrix M is said to be d-disjunct if the union of any set of d columns does not contain any other column as a subset.*

Lemma 6.14. *An incidence matrix M is d-disjunct if, and only if, given any column $\mathbf{c}_j$ and any set of d other columns, there is at least one row in which $\mathbf{c}_j$ has a one and all the d columns have a zero.*

Lemma 6.15. *If an incidence matrix M is d-disjunct then M is d'-disjunct and d'-separable for all $d' \leq d$.*

Lemma 6.16. *If an incidence matrix M is $\overline{d}$-separable then M is d'-disjunct for all $d' < d$.*

Theorem 6.17. *The incidence matrix M of a route design is d-disjunct if and only if, for any distribution of at most d malicious hosts, the union of all negative rows of M (i.e. rows that yield a negative outcome) contains all the negative columns of M (i.e. columns that correspond to non-defective hosts).*

Theorem 6.17 implies that the decoding algorithm of a d-disjunct design has complexity linear in n, since items (columns) not appearing in negative rows are defective (positive). In other words, in a d-disjunct design, if a column vector is contained in the outcome vector, then this column vector corresponds to a defective host.

Corollary 6.18. *The route design of a d-disjunct matrix is a $\overline{d}$-classifier. Given hosts $\mathbf{c}_j$ and the route design outcome $\mathbf{a}$, the binary vector $\boldsymbol{\delta}$ of all malicious hosts, $|\boldsymbol{\delta}| \leq d$, is determined by the following decoding algorithm:*

$$\boldsymbol{\delta} = \{j : \mathbf{c}_j \subseteq \mathbf{a}\}\,.$$

6.4.2 A simple block design construction

There are numerous constructions for d-disjunct matrices [46, 190]. In this chapter we consider constructions based on block designs (see §2.6.5.1). We focus specifically on 2-designs (see Definition 2.12) for the following reasons.

- If we use a block design as a route design, taking blocks as hosts and elements of V as routes, then the security requirement $|R_i \cap R_j| \leq 1$ (in Problem 6.6) is automatically satisfied by a 2-$(v, k, 1)$ design.
- It can be shown [9, 54] that for a small number of defectives, more precisely if $d \leq 2$, there are cases where 2-designs are *optimal pooling designs* in that they identify the defectives using a minimum number of tests. Such optimal pooling designs are also of interest as route designs since they maximise the number of hosts contained in each test.
- There are many known construction methods for 2-designs (see, for example, [15, 35]).

Theorem 6.19. *An incidence matrix of a t-$(v, k, 1)$ design, where the blocks correspond to columns and the elements to rows, is $(q-1)$-disjunct, where:*

$$q = \lceil \frac{k}{t-1} \rceil . \tag{6.4.1}$$

Proof. By definition of a t-design and since $\lambda = 1$, two columns intersect in at most $t-1$ rows. The weight of each column is $|B| = k$. Hence, it takes the union of at least $q = \lceil \frac{k}{t-1} \rceil$ columns to cover another column [105, Theorem 6]. The result follows. □

From Theorem 6.19 it follows immediately that a 2-$(v, k, 1)$ design is a d-disjunct route design, where:

$$d = k - 1 . \tag{6.4.2}$$

This gives a simple means of constructing sets of visited hosts for spy agents. If we assume the use of a 2-$(v, k, 1)$ design, then the optimisation problem given in Problem 6.6 becomes rather simpler. This is because there is much less freedom to choose the parameters of the route design (essentially there are only two 'degrees of freedom'). Note, however, that the reduced problem may exclude some optimal solutions.

Problem 6.20. *Find a* 2-$(v, k, 1)$ *design in the following scenarios:*

1. *(Security): Given the number of target hosts* n $[= b]$*, find a design that maximises the cardinality of each route* r*.*

2. *(Efficiency): Given the number of spy agents* v*, find the maximum number of target hosts* b *that can be tested. This is achieved when* k *is minimised, since, from* (2.6.1a) *and* (2.6.1b)*,* $b = \frac{v(v-1)}{k(k-1)}$*.*

In Problem 6.20 the choice of a minimum d $[= k - 1]$, gives a 2-design which is optimal both from a security and an efficiency point of view. This follows since a larger number of blocks (i.e. target platforms) implies a larger number of blocks containing an element (i.e. larger route cardinality). However, the choice of d is critical. An underestimate for the maximum number of malicious hosts could mean that malicious hosts will not be identified, whereas an overestimate for the maximum number of malicious hosts will reduce the efficiency and security of the route design.

6.4.3 Further examples

The finite projective plane of order two is known as the Fano plane, and is a 2-design with $v = b = 7$, $k = r = 3$ and $\lambda = 1$. The corresponding spy agent system thus has seven hosts, seven tests, three hosts per test (and three tests

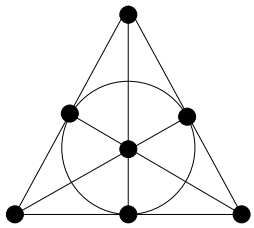


Figure 6.1: Fano plane.

Table 6.2: 2-$(7, 3, 1)$ incidence matrix.

	B_1	B_2	B_3	B_4	B_5	B_6	B_7
v_1	1	1	1	0	0	0	0
v_2	1	0	0	1	1	0	0
v_3	1	0	0	0	0	1	1
v_4	0	1	0	1	0	1	0
v_5	0	1	0	0	1	0	1
v_6	0	0	1	1	0	0	1
v_7	0	0	1	0	1	1	0

per host). The Fano plane is shown diagrammatically in Figure 6.1. Note that each point has three lines on it and each line contains three points.

An incidence matrix for the Fano Plane is given in Table 6.2. This design has the following seven blocks: $(V, \mathcal{B}) : \mathcal{B} = \{B_1, B_2, B_3, B_4, B_5, B_6, B_7\}$, $B_1 = \{v_1, v_2, v_3\}$, $B_2 = \{v_1, v_4, v_5\}$, $B_3 = \{v_1, v_6, v_7\}$, $B_4 = \{v_2, v_4, v_6\}$, $B_5 = \{v_2, v_5, v_7\}$, $B_6 = \{v_3, v_4, v_7\}$ and $B_7 = \{v_3, v_5, v_6\}$.

It is trivial to verify that this matrix is 2-disjunct but not 3-disjunct (since $d = k - 1 = 2$). Hence, this design will successfully identify two defectives but not three.

Example 6.21. Suppose that the first two hosts in the matrix in Table 6.2 are malicious. The outcome vector of this matrix is $\mathbf{a} = \{1, 1, 1, 1, 1, 0, 0\}$. From Corollary 6.18, the defectives (i.e. the hosts not appearing in all the negative tests, i.e. $\mathbf{r}_6$ and $\mathbf{r}_7$) are $\mathbf{c}_1$ and $\mathbf{c}_2$. However, if the first three hosts are defective, then the outcome vector is $\mathbf{a} = \{1, 1, 1, 1, 1, 1, 1\}$. The decoding algorithm

given in Corollary 6.18 suggests that all hosts are defective, contradicting the implicit assumption that there are most two defectives. It is thus impossible to decide which hosts are defective. The problem of dealing with inconsistent results is further discussed in Chapter 9.

The process of finding suitable classifier designs may be further simplified by choosing a minimum spy agent route length. This is shown in the example that follows.

Example 6.22. Suppose that we require each spy route to contain at least five hosts and $d = 2$. Hence:

$$r \geq 5 \overset{(2.6.1b)}{\Longrightarrow} \frac{(v-1)}{k-1} \geq 5 \Rightarrow v \geq 5k - 4 \overset{(6.4.2)}{\Longrightarrow} v \geq 11$$

One set of 2-design parameters satisfying this inequality has $(v, b, r, k, \lambda) = (15, 35, 7, 3, 1)$. There are known to be at least 80 pair-wise nonisomorphic 2-designs with these parameters [35], which could, for example, be utilised for multiple re-evaluations of the same set of hosts.

6.5 Conclusions

Remote security evaluations are inherently unreliable. Malicious hosts may selectively misbehave depending on whether or not they believe they are being evaluated. In this chapter we have defined a way of identifying the malicious hosts in a hostile network, while maximising the chance that they will misbehave. This has been achieved by applying combinatorial group testing theory to the spy agent paradigm. A simple class of block designs is proposed for use in constructing spy agent routes with the desired properties.

The mathematical model of agent route design that we have presented gives rise to further challenges, including the following.

- How might the rather strong assumptions made in §6.3.1 be relaxed? More specifically:
  a. What if malicious nodes behave badly in a probabilistic fashion, e.g. if they only misbehave when the incentive for them to do so is maximised?
  b. What if the spy agent route design is not optimal?
  c. Can the requirements for maximum route size and minimum route intersection size be justified in a practical scenario?
  d. What other optimality criteria could usefully be considered?
- What practical scenarios can be devised, and what aspects of host trustworthiness might be evaluated?
- How can spy agents be constructed, what data could they carry, and what operations might they run remotely, e.g. in an e-commerce scenario?
- How could spy agent network scenarios be compromised by colluding malicious hosts, and how might this be addressed?

Some of these issues are addressed in other chapters of this thesis, or in the prior art. More specifically, we make the following observations.

- Inconsistent behaviour could be addressed by using other GT designs such as error-tolerant GT designs (see §2.6.3). This scenario is not studied further in this thesis as the adaptation of known results (see, for example, Balding and Torney [9]) is straightforward.
- A different class of combinatorial design that identifies malicious hosts exhibiting other types of malicious behaviour, including collusion, is studied in Chapter 7.

- Chapter 8 describes alternative criteria and spy agent schemes using sequential GT.
- Chapter 9 discusses a method that can be used to evaluate the credibility of spy agent system results.
- Chapter 10 discusses spy agent applications and scenarios in which the requirements and assumptions specified earlier in the thesis are justified.

These issues are also possible topics for future research. This is further discussed in Chapter 11.

"Good composition is like a suspension bridge—each line adds strength and takes none away"

Robert Henri

7

Complex spy agent group testing

Contents

7.1 Synopsis

This chapter extends the route design methodology given in Chapter 6. The main goal is to enhance the properties of the set of spy agent routes to make the spy agent system resilient to malicious hosts that collude with other malicious hosts, or use information gained from observing the agents before deciding whether or not to misbehave. The new model gives rise to a novel group testing problem in which some defectives yield a positive test outcome only when other defectives are included in the test. We analyse this problem and give a range of new results, including an efficient means of analysing the results obtained when using a particular class of spy agent route design.

The remainder of this chapter is organised as follows. The problem of group testing for hosts that are collectively malicious is formulated in §7.2. Aspects of the theory of combinatorial 'group testing for complexes' are discussed and extended in §7.3. Building on the previous discussions, in §7.4 the problem of spy agent testing of collectively malicious hosts is linked to the group testing for complexes problem. A rank-2 host classification algorithm is proposed and analysed in §7.5, and conclusions are drawn in §7.6.

Aspects of the work described in this chapter have been published in [99].

7.2 Problem formulation

7.2.1 Behaviour model for malicious hosts

In Chapter 6 it was assumed that an untrustworthy spy agent host will always perform an act that will enable the sender of the agent to detect that it has visited a malicious platform. That is, we have assumed that a spy agent that visits a malicious host will always yield a positive outcome. In this chapter we

relax this rather strong requirement. That is, we now suppose that a malicious host may not always exhibit characteristic (malicious) behaviour, even if the host does not necessarily know that a visiting agent is a spy agent (i.e. even if the spy agent security requirements are met). As previously, in this chapter we treat a spy agent route as the (unordered) set of hosts visited by the agent.

In general, malicious host behaviour can be either stochastic (where a malicious host behaves randomly) or deterministic. In this chapter we focus on a particular type of deterministic malicious behaviour. (Results from the prior art applying to route designs that tolerate errors due to stochastic behaviour were briefly discussed in §6.5.)

More specifically, in this chapter we extend the theory of Chapter 6 by considering the case where a malicious host will only abuse a visiting spy agent if it either colludes with, or at least recognises, one or more other malicious hosts within the spy agent route. The idea is that a (cautious) malicious host might employ such a selective misbehaviour model in order to escape detection by a spy agent system (this issue was discussed in general in §1.3). The following two types of selective malicious host behaviour are considered here.

- **Model 1**. An 'individually malicious' host will violate all visited spy agents (in a detectable manner) provided that the lengths of the agent routes are above a certain threshold. (Ideally, route lengths should be maximised, as discussed in §6.3.1—see also the first item listed in Problem 6.6). We refer to such malicious hosts as `Type-1` hosts.
- **Model 2**. A 'collectively malicious' host will violate a spy agent if and only if (a) the agent route length is above a certain threshold (in subsequent discussions we will assume that this threshold is fixed across

the host population), and (b) there are at least $e-1$, ($e > 1$), other hosts in the agent route that are known by the host to be malicious. Otherwise, the collectively malicious host will not violate the agent data. We refer to such malicious hosts as `Type-`e hosts.

The discussion in the previous chapter applies to the case where all hosts will behave according to the first model given above. In this chapter we generalise our discussion to also cover hosts of the second type.

The second behaviour model fits a scenario in which malicious hosts collude to manipulate visiting agents. However, this model is also applicable when hosts decide whether or not to misbehave depending on the information they hold about the behaviour of other hosts in a spy agent route. The rationale for a malicious host behaving in such a manner is that a malicious host may plausibly be able to deny abuse of a spy agent if the agent in question also visits one or more other hosts known to be malicious. The route designs discussed in Chapter 6 could fail to reliably identify malicious hosts exhibiting such selective malicious behaviour. In this chapter we consider how to identity malicious hosts behaving according to this more complex model.

Of course, the second host misbehaviour model could apply in other scenarios. For example, it might apply when spy agents are used to analyse truncation attacks involving colluding hosts (see §2.4.2.1.3). Such applications are further discussed in Chapter 10.

It is also important to note that, in general, the information that malicious hosts hold about other hosts may not be correct; i.e. the fact that a host believes another host to be malicious (or not) does not necessarily mean that the other host is actually malicious (or not). However, in this chapter we make the simplifying assumption that a `Type-`e ($e > 1$) host will always know

the identities of (and may potentially collude with) other malicious hosts. For example, malicious hosts might use spy agent technology to deploy a set of malicious agents that aim to help identify peer malicious hosts.

Unlike in Problem 6.6, we do not assume here that the number of hosts common to two spy agent routes will affect malicious host behaviour. This is likely to be the case when many agents (both spy agents and 'normal' e-commence agents) visit a host over a short period of time. In this case we consider that an overlap between the route sets of visiting agents is likely to be a typical, 'non-suspicious' event.

We formalise the new model in the following way.

Definition 7.1. *Suppose $e \geq 1$. When processing a mobile agent whose route contains $d \geq 0$ malicious hosts, a* `Type-`*e (malicious) host will behave as follows:*

- *if $d \geq e$ then the host will handle the agent in such a way that it returns a positive result; and*
- *if $d < e$ then the host will handle the agent in such a way that a negative result will ensue unless another host abuses the agent.*

By convention, we say that a non-malicious host is a `Type-0` host.

In summary, we assume that each deployment of a spy agent will yield a result of:

- 'positive' if, for some $e \geq 1$, its route contains at least one `Type-`e host and at least $e - 1$ other malicious hosts, and
- 'negative' in all other cases.

The problem of efficiently identifying `Type-`e malicious hosts in such a scenario is a generalisation of the classic GT problem, which reduces to the classic problem in the case $e = 1$.

7.2.2 Notation

Using the notation introduced in §6.3.3.1, we treat a spy agent route design as an incidence structure $\mathcal{H} = (S, \mathcal{R})$, where S is the set of n target hosts and $\mathcal{R}$ is a set of routes, i.e. subsets of S. We also let $M = (m_{ij})$ be the incidence matrix of $\mathcal{H}$, and we refer to a route $\mathbf{r}_i$ and a host $\mathbf{c}_j$.

We suppose that each defective item is of `Type-`ε, for some $\varepsilon \geq 1$. The set of all defective items can thus be partitioned into the sets $\boldsymbol{\delta}_\varepsilon$, $\varepsilon \geq 1$, where

$$\boldsymbol{\delta}_\varepsilon = \{\text{hosts } \boldsymbol{c}_j \,|\, \boldsymbol{c}_j \text{ is } \texttt{Type-}\varepsilon\} \ . \tag{7.2.1}$$

The set of all defectives (i.e. misbehaving hosts), which we denote by Δ, trivially satisfies $\Delta = \bigcup_{\varepsilon \geq 1} \boldsymbol{\delta}_\varepsilon$. From Definition 7.1 it follows that $|\boldsymbol{\delta}_i \cap \boldsymbol{\delta}_j| = 0$ for $i \neq j$, i.e. a defective item can only have one `Type` of behaviour.

We next define the *outcome vector* for a route design when applied to hosts conforming to the model introduced in this chapter.

Definition 7.2. *Given a particular set S of hosts (some of which may be defective), the outcome vector of a route design $(S, \mathcal{R})$ is the vector $\boldsymbol{a} = \big(a_1, a_2, \ldots, a_{|\mathcal{R}|}\big)$, where $a_i = 1$ if the route $\boldsymbol{r}_i \in \mathcal{R}$ contains at least one `Type-`ε host and $\varepsilon - 1$ other malicious hosts, for at least one ε ($1 \leq \varepsilon \leq e$), and $a_i = 0$ otherwise.*

Given the above assumptions about the behaviour of malicious nodes (defectives), the outcome vector indicates the outcomes of the routes (spy agents) when used on a specific target host set, where 1 and 0 correspond to positive

Table 7.1: A simple route design for four hosts.

	$\boldsymbol{c}_1$	$\boldsymbol{c}_2$	$\boldsymbol{c}_3$	$\boldsymbol{c}_4$
$\boldsymbol{r}_1$	1	1	0	0
$\boldsymbol{r}_2$	1	0	1	0
$\boldsymbol{r}_3$	1	0	0	1
$\boldsymbol{r}_4$	0	1	1	0
$\boldsymbol{r}_5$	0	1	0	1
$\boldsymbol{r}_6$	0	0	1	1

and negative outcomes, respectively. For this reason we also refer below to the *outcome set*, meaning the set of routes (spy agents) yielding a positive result.

Example 7.3. Consider the incidence matrix in Table 7.1. In this matrix a set of six spy agents is used to test four hosts, where each pair of hosts is visited by a unique agent. Suppose that there are two `Type-2` malicious hosts, where $\Delta = \boldsymbol{\delta}_2 = \{\boldsymbol{c}_1, \boldsymbol{c}_4\}$. Clearly, the only agent containing $\boldsymbol{\delta}_2$ is $\boldsymbol{r}_3$. Hence, in this case the outcome vector is $\boldsymbol{a} = (0, 0, 1, 0, 0, 0)$. If the spy agent system user knows that every malicious host has `Type` at most 2, then the outcome of the route design enables the identification of $\boldsymbol{c}_1$ and $\boldsymbol{c}_4$ as malicious.

In order to address the general case, i.e. where there are mixed `Types` of malicious hosts, we need the following definitions.

Definition 7.4. *If the outcome vector of a route design can be used to distinguish between honest and malicious hosts regardless of how they are distributed, as long as there are exactly d malicious nodes and the `Type` of each malicious node is exactly e, then the route design is called a* (d, e)-classifier.

Definition 7.5. *If the outcome vector of a route design can be used to distinguish between honest and malicious hosts regardless of how they are distributed, as long as there are at most d malicious nodes and the `Type` of each malicious node is exactly e, then the route design is called a* $(\overline{d}, e)$-classifier.

Definition 7.6. *If the outcome vector of a route design can be used to distinguish between honest and malicious hosts regardless of how they are distributed, as long as there are exactly d malicious nodes and the* `Type` *of each malicious node is at most e, then the route design is called a* $(d, \overline{e})$-classifier.

Definition 7.7. *If the outcome vector of a route design can be used to distinguish between honest and malicious hosts regardless of how they are distributed, as long as there are at most d malicious nodes and the* `Type` *of each malicious node is at most e, then the route design is called a* $(\overline{d}, \overline{e})$-classifier.

In the context of this chapter, a classifier (route) design, or simply a classifier, refers to a (route) design that is either a $(\overline{d}, \overline{e})$-*classifier*, a $(\overline{d}, e)$-*classifier*, a $(d, \overline{e})$-*classifier*, or a (d, e)-*classifier*, unless otherwise stated.

The following three results are immediate from Definitions 6.2, 6.3, 7.4, 7.5, 7.6 and 7.7.

Lemma 7.8. *A route design is a $(\overline{d}, 1)$-classifier if and only if it is a $\overline{d}$-classifier.*

Lemma 7.9. *A route design is a $(d, 1)$-classifier if and only if it is a d-classifier.*

Lemma 7.10. *A route design is a $(\overline{d}, \overline{e})$-classifier if and only if it is a (d', e')-classifier for every pair (d', e') satisfying $1 \leq d' \leq d$ and $1 \leq e' \leq e$.*

As in Problem 6.20, we are interested in constructing classifier route designs for a range of possible values of d and e. Since we claim that routes which contain a large number of hosts are less likely to lead to suspicion, we are interested in route designs with a large minimum route size. Ideally, we also wish to identify malicious hosts with the minimum effort, and therefore route

designs which minimise the number of routes (and hence agents) are also to be preferred.

7.3 Group testing for complexes (GTC)

7.3.1 The notion of complex defective

Before we consider the construction and use of classifiers, we need to adapt and extend certain definitions from the theory of GTC, introduced in §2.6.4.

In GTC we wish to identify items that are 'collectively' positive, in contrast with GT theory where items are 'individually' positive. To proceed, we first need to introduce some further notation.

Definition 7.11. *Let C be a subset of the columns of an incidence matrix M of a route design, where $|C| = \varepsilon$. We call C an ε-complex, and we denote the intersection of all the columns in C by $\cap C$, where*

$$\cap C = \bigcap_{\boldsymbol{c}_i \in C} \boldsymbol{c}_i \ . \tag{7.3.1}$$

The following Lemma follows trivially from the Definition.

Lemma 7.12. *If the intersection $\cap C$ of an ε-complex C contains a route $\boldsymbol{r}_i$ then this route contains all the columns in C.*

When a route contains all the columns of a complex, we also say that the complex appears in the route.

The notion of a complex is useful in the context of GTC designs because the presence of a single defective column in a row will not necessarily cause the associated test to return a positive result; instead it will be necessary for all the elements of a complex of a certain size to be present in a row and for them all to be defective in order for the test to return a positive result. That is,

for any tested set of items, there is an associated set of complexes (i.e. sets of subsets of the columns) with the property that a row will give a positive result if and only if it contains all the elements of at least one of these complexes. We call this a *set of defective complexes*.

Defective complexes were introduced in [180], where it was assumed that "for obvious reasons, no positive subset may include any other positive subset". Macula et al. [112] has further studied GTC in the case where all defective complexes are k-sets, i.e. they all have cardinality k. In this case, each complex is called a k-complex. In the equivalent hypergraph testing problem (see §2.6.4), the set of defective complexes corresponds to a set of defective (hyper)edges, and an assumption is made that no defective edge contains another defective edge [63]. Similar language is used by Chen et al. [30], where the theory of hypergraph testing (and GTC) is linked with the theory of cover-free families.

It would appear that the notion of a defective complex is rather loosely defined in the literature. GTC is a relatively young area of research, with applications mainly in the field of DNA testing. We therefore attempt below to provide a more rigorous discussion. We first observe that, as we have informally defined the notion above, there may be more than one set of defective complexes. For example, suppose that $\boldsymbol{c}_1$ is the only defective host (where $\boldsymbol{c}_1$ is a `Type-1` host) in a set of n hosts $\{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n\}$. Then clearly $\mathcal{D} = \{\{\boldsymbol{c}_1\}\}$ is a set of defective complexes. However, it is not hard to see that $\mathcal{D} = \{\{\boldsymbol{c}_1\}, \{\boldsymbol{c}_1, \boldsymbol{c}_2\}\}$ is also a set of defective complexes. In order to make $\mathcal{D}$ unique, we first make the following definition.

Definition 7.13. *Given a set of hosts, a set of defective complexes is a collection $\mathcal{D}$ of subsets of hosts with the property that:*

1. *a test chosen from the set of all possible tests (i.e. all possible routes) will give a positive result if and only if it contains one of the members of $\mathcal{D}$;*

2. *the previous property does not hold for any proper subset of $\mathcal{D}$.*

The following remark follows trivially.

Remark 7.14. *Suppose a route design is applied to a set of hosts for which $\{D_1, D_2, \ldots, D_n\}$ is a set of defective complexes. Then the outcome vector of the route design is the union of $\cap D_i$, $1 \leq i \leq n$.*

We note that the second property, i.e. the minimality condition, in Definition 7.13 is consistent with the literature. This is because, in the context of DNA testing, a set of DNA molecules is 'defective' if and only if all the molecules in the set are required to cause a disease.

Example 7.15. Suppose that, as above, $\boldsymbol{c}_1$ is the only defective host (where $\boldsymbol{c}_1$ is a `Type-1` host) in a set of n hosts $\{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n\}$. Then, although $\mathcal{D} = \{\{\boldsymbol{c}_1\}\}$ is a set of defective complexes, by Definition 7.13 $\mathcal{D}' = \{\{\boldsymbol{c}_1\}, \{\boldsymbol{c}_1, \boldsymbol{c}_2\}\}$ is not a set of defective complexes.

We now show that, under Definition 7.13, the set of defective complexes is unique.

Lemma 7.16. *All members of a set of defective complexes are mutually non-inclusive.*

Proof. Let $\mathcal{D}$ be a set of defective complexes and suppose $D_1 \subset D_2$, where $D_1, D_2 \in \mathcal{D}$. Any test that contains D_2 will also contain D_1. Hence, any test that contains a member of $\mathcal{D}$ will also contain a member of $\mathcal{D}' = \mathcal{D} - \{D_2\}$, which contradicts Definition 7.13. □

Lemma 7.17. *If $\mathcal{D}$ is a set of defective complexes then $\mathcal{D}$ is unique.*

Proof. Suppose that $\mathcal{D}$ is not unique. Let $\mathcal{D}$ and $\mathcal{D}'$ be distinct sets of defective complexes, and suppose (without loss of generality) that $D_1 \in \mathcal{D}'$ and $D_1 \notin \mathcal{D}$. Consider the route containing only the elements of D_1. This route gives a positive result because D_1 is contained in $\mathcal{D}'$. Hence, this route contains at least one defective complex $D_2 \in \mathcal{D}$, where $D_2 \subset D_1$. Now consider the route containing only the elements of D_2. This route gives a positive result since D_2 is contained in $\mathcal{D}$. Hence, this route contains at least one defective complex $D_3 \in \mathcal{D}'$, where $D_3 \subseteq D_2$. That is, $\mathcal{D}'$ contains D_3 and D_1, and $D_3 \subset D_1$. This contradicts Lemma 7.16, and the result follows. □

From Lemma 7.17 if follows that the unique set of defective complexes in Example 7.15 is $\mathcal{D} = \{\{\mathbf{c}_1\}\}$.

7.3.2 Property of GTC designs

The primary goal of GTC theory is to find a design which enables the set of defective complexes to be identified. That is, we wish to find a GTC design which has the property that the outcome vectors for any two distinct sets of defective complexes will always be different.

With this goal in mind, we next adapt and extend the notions of separable and disjunct GTCs, as given by Du and Hwang [46, Chapter 6].

Definition 7.18. *The* rank *of a set of complexes $\mathcal{C} = \{C_1, C_2, \ldots, C_{|\mathcal{C}|}\}$ is the size of the largest complex, i.e.* $\max_{1 \leq i \leq |\mathcal{C}|} |C_i|$.

Definition 7.19. *A set $\mathcal{C}$ of complexes is said to have uniform rank if all the complexes in $\mathcal{C}$ have the same size.*

Definition 7.20. *A route design is said to be (d, e)-separable if, given any two distinct sets of defective complexes of size d and rank e, it yields distinct outcome sets.*

Definition 7.21. *A route design is said to be $(\overline{d}, e)$-separable if, given any two distinct sets of defective complexes of size at most d and rank e, it yields distinct outcome sets.*

Definition 7.22. *A route design is said to be $(d, \overline{e})$-separable if, given any two distinct sets of defective complexes of size d and rank at most e, it yields distinct outcome sets.*

Definition 7.23. *A route design is said to be $(\overline{d}, \overline{e})$-separable if, given any two distinct sets of defective complexes of size at most d and rank at most e, it yields distinct outcome sets.*

Definition 7.24. *A route design is said to be (d, e^*)-separable if, given any two distinct sets of defective complexes of size d and uniform rank e, it yields distinct outcome sets.*

Definition 7.25. *A route design is said to be $(\overline{d}, e^*)$-separable if, given any two distinct sets of defective complexes of size at most d and uniform rank e, it yields distinct outcome sets.*

Definition 7.26. *A route design is said to be (d, e)-disjunct if, given any set of $d + 1$ mutually non-inclusive complexes $\{C_0, C_1, C_2, \ldots, C_d\}$ with rank e,*

$$\cap C_0 \nsubseteq \bigcup_{i=1}^{d} (\cap C_i) \quad . \tag{7.3.2}$$

Definition 7.27. *A route design is said to be $(d, \overline{e})$-disjunct if, given any set of $d + 1$ mutually non-inclusive complexes $\{C_0, C_1, C_2, \ldots, C_d\}$ with rank at most e, Equation 7.3.2 applies.*

Definition 7.28. *A route design is said to be (d, e^*)-disjunct if, given any set of $d+1$ mutually non-inclusive complexes $\{C_0, C_1, C_2, \ldots, C_d\}$ with uniform rank e, Equation 7.3.2 applies.*

The following three lemmas follow immediately from the above three definitions.

Lemma 7.29. *A route design that is (d, e)-disjunct is (b, e)-disjunct for $1 \leq b \leq d$.*

Lemma 7.30. *A route design that is $(d, \overline{e})$-disjunct is $(b, \overline{e})$-disjunct for $1 \leq b \leq d$.*

Lemma 7.31. *A route design that is (d, e^*)-disjunct is (b, e^*)-disjunct for $1 \leq b \leq d$.*

The following six theorems are from Du and Hwang [46, Chapter 6], restated in our notation.

Theorem 7.32. *A route design that is (d, e)-disjunct is $(\overline{d}, e)$-separable.*

Theorem 7.33. *A route design that is $(d, \overline{e})$-disjunct is $(\overline{d}, \overline{e})$-separable.*

Theorem 7.34. *A route design that is (d, e^*)-disjunct is $(\overline{d}, e^*)$-separable.*

Theorem 7.35. *A route design that is (d, e)-separable is $(d-1, e)$-disjunct.*

Theorem 7.36. *A route design that is $(d, \overline{e})$-separable is $(d-1, \overline{e})$-disjunct.*

Theorem 7.37. *A route design that is (d, e^*)-separable is $(d-1, e^*)$-disjunct.*

Example 7.38. Consider the route design $\mathcal{H}$ given in Table 7.1. Each of the six routes of this design contains a distinct pair of hosts from the population of four hosts. The following properties hold.

- $\mathcal{H}$ is $(1,1)$-disjunct because, given any column (i.e. a 1-complex), no other column is contained in this column.
- $\mathcal{H}$ is $(2,1)$-disjunct for the following reason. The union of any two columns (1-complexes) will include all but one row. This row will always be contained in the remaining two columns. It follows that the union of any two columns will not contain either of the other two columns.
- $\mathcal{H}$ is $(1,2)$-disjunct. To establish this we need to show that the intersection of a complex of size at most two cannot be contained in the intersection of a distinct complex of size at most two. Consider a complex containing either one or two columns. If it contains one column, then this column is not contained in any other column and is thus not contained in the intersection of any two other columns. If the complex contains two columns, then the intersection of these two columns is not contained in either of the other two columns and is thus not contained in the intersection of any pair of columns (distinct from the complex itself).
- $\mathcal{H}$ is $(2,2)$-disjunct. To establish this we need to show that the intersection of columns in a complex of size at most 2 is not contained in the union of the intersection of any two complexes of size at most 2. Clearly, if we can show that the intersection of columns in a complex of size exactly 2 is not contained in the union of the intersection of two complexes of size exactly 1 (not contained in the complex of size 2) then the result will follow. However, this follows immediately since the intersection of two columns contains a unique row, which is not contained in either of the other two columns.

- $\mathcal{H}$ is $(2, \overline{2})$-disjunct. This follows immediately, given that $\mathcal{H}$ is $(2, 1)$-disjunct and $(2, 2)$-disjunct.
- $\mathcal{H}$ is $(\overline{2}, \overline{2})$-separable. This follows immediately from the fact that $\mathcal{H}$ is $(2, \overline{2})$-disjunct (see above) and Theorem 7.33.
- $\mathcal{H}$ is neither $(3, 2)$-disjunct nor $(3, 2)$-separable because the union of any three distinct columns contains all rows.
- $\mathcal{H}$ is $(5, 2^*)$-disjunct, because the intersection of any 2-complex contains only one row, which is distinct from the single row contained in the intersection of each of the five other 2-complexes.
- $\mathcal{H}$ is $(\overline{5}, 2^*)$-separable. This follows immediately from the fact that $\mathcal{H}$ is $(5, 2^*)$-disjunct (see above) and Theorem 7.34.

The disjunct and separable properties were originally studied in two contexts. First, in graph testing, a hypergraph, i.e. a generalisation of the notion of a graph in which an edge can connect any number of vertices, is tested in order to identify a defective subgraph [45, Chapter 10]. Second, in the DNA complex model, a set of molecules is tested to identify diseases, where each disease corresponds to a subset of molecules [112]. More recently, the GTC problem has been connected to three further application domains, namely secure key distribution [120], binary superimposed codes [47] and cover-free families [190]. The importance of these connections is that known results across all these problems can be used in the context of GTC. We adapt some existing results to our setting below.

Definition 7.39. (Mitchell and Piper [120]) *A route design is a (d, e)-KDPif, for any set of $d + e$ distinct columns $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_{d+e}$,*

$$\bigcap_{i=1}^{e} \boldsymbol{c}_i \nsubseteq \bigcup_{i=e+1}^{e+d} \boldsymbol{c}_i \ . \tag{7.3.3}$$

Note that a (d, e)-KDP is the same as a (d, e)-superimposed code and a (d, e)-cover-free family [30].

Theorem 7.40. (Chen et al. [30]) *A route design is (d, e)-disjunct if and only if the route design is a (d, e)-KDP.*

The next lemma helps identify a set of defective complexes of uniform rank given the output of a disjunct route design.

Lemma 7.41. *Let S be a set of hosts, let $\mathcal{D}$ be the set of defective complexes for S, which we assume have uniform rank e, and let $\mathcal{E}$ be the set of all possible e-subsets (e-complexes) of S, for some $e \geq 1$. Suppose that $|\mathcal{D}| = d$, and that a route design gives output vector $\boldsymbol{a}$. If the route design is (d, e^*)-disjunct, then all the complexes in $\mathcal{E} - \mathcal{D}$ appear in a route $\boldsymbol{r}_i$ that tests negative.*

We can use this lemma to give a simple decoding algorithm, as follows.

Algorithm 7.42. *Suppose a (d, e^*)-disjunct route design has outcome* **a** *when applied to a specific set of hosts S and suppose that the set $\mathcal{D}$ of defective complexes for S has cardinality d and uniform rank e. Then, $\mathcal{D}$ can be determined using the following decoding algorithm:*

- **Step 1***: Create the set $\mathcal{E} = \{E_1, E_2, \ldots, E_{|\mathcal{E}|}\}$ containing all possible e-complexes.*
- **Step 2***: Put $\mathcal{D} = \{E_i : \cap E_i \subseteq \boldsymbol{a}\}$.*

We next generalise Lemma 7.41 to the case where the rank of the set of defective complexes is not uniform. We need this generalisation since (as shown below) defective hosts of non-uniform `Type` give rise to defective complexes of varying sizes.

Theorem 7.43. *Let S be a set of hosts, let $\mathcal{D}$ be the set of defective complexes for S, which we suppose has rank e, and let $\mathcal{E}$ be the set of all possible ε-subsets (ε-complexes) of S, for every ε, ($1 \leq \varepsilon \leq e$). Suppose that $|\mathcal{D}| \leq d$, and that a route design gives output vector $\boldsymbol{a}$ when applied to S. Suppose that $\mathcal{D}'$ is the superset of $\mathcal{D}$ containing every superset (within S) of a defective complex. If the route design is (d, e)-disjunct, then all complexes in $\mathcal{E} - \mathcal{D}'$ appear in a route $\boldsymbol{r}_i$ that tests negative.*

Proof. Suppose that the route design is (d, e)-disjunct. Then, by definition, given any $d + 1$ mutually non-inclusive complexes of cardinality at most e, there is at least one row contained in the intersection of the first complex that is not contained in the intersections of any of the other complexes. Choose any complex $E_i \in (\mathcal{E} - \mathcal{D}')$, $|E_i| \leq e$. We first show that all the complexes in the set $\{E_i\} \cup \mathcal{D}$ are mutually non-inclusive. This follows because all the elements of $\mathcal{D}$ are mutually non-inclusive (by definition) and E_i is not a superset of a defective complex. It follows that there is a row, say $\boldsymbol{r}_j$, that is contained in $\cap E_i$ and is not contained is the intersections of any of the d complexes in $\mathcal{D}$. Hence, $a_j = 0$ (where a_j is the bit in $\boldsymbol{a}$ corresponding to $\boldsymbol{r}_j$), and the result follows. □

We can use Theorem 7.43 to generalise the decoding Algorithm 7.42 in the following way.

Algorithm 7.44. *Suppose that a (d, e)-disjunct route design has outcome* $\mathbf{a}$ *and the set $\mathcal{D}$ of defective complexes has cardinality at most d and rank e. Then, $\mathcal{D}$ can be determined using the following decoding algorithm.*

- **Step 1***: Create the set $\mathcal{E} = \{E_1, E_2, \ldots, E_{|\mathcal{E}|}\}$ containing all possible ε-complexes, for every ε, $(1 \leq \varepsilon \leq e)$.*
- **Step 2***: Put $\mathcal{G} = \{E_i : \cap E_i \subseteq \boldsymbol{a}\}$.*
- **Step 3***: Put $\mathcal{D} = \{G_i : G_i \not\supset G_j, \text{ where } G_i, G_j \in \mathcal{G}, \text{ for all } i \neq j\}$.*

Proof of correctness. Since all the complexes that appear in a route giving a negative result are non-malicious, each of the complexes in $\mathcal{G}$ contains at least one defective complex. Step 3 guarantees that the members of $\mathcal{D}$ are non-inclusive, which is consistent with the minimality property in Definition 7.13. □

Example 7.45. Consider again the $(2, \overline{2})$-disjunct route design in Table 7.1, in which six spy agents visit a set of four hosts, and each spy agent visits a different pair of hosts. Suppose that this spy agent route design is applied in four different scenarios where the sets of defective complexes are as follows.

$$\begin{aligned} \mathcal{D} &= \{\{\boldsymbol{c}_1\}, \{\boldsymbol{c}_2\}\} \\ \mathcal{D}' &= \{\{\boldsymbol{c}_1\}, \{\boldsymbol{c}_2, \boldsymbol{c}_3\}\} \\ \mathcal{D}'' &= \{\{\boldsymbol{c}_1, \boldsymbol{c}_2\}, \{\boldsymbol{c}_2, \boldsymbol{c}_3\}\} \\ \mathcal{D}''' &= \{\{\boldsymbol{c}_1, \boldsymbol{c}_2\}, \{\boldsymbol{c}_3, \boldsymbol{c}_4\}\} \end{aligned}$$

In each scenario, there are at most two defective complexes and each defective complex contains at most two hosts. The outcomes $\boldsymbol{a}$, $\boldsymbol{a}'$, $\boldsymbol{a}''$, $\boldsymbol{a}'''$ of the corresponding designs for the four scenarios are shown in Table 7.2. With the (a priori) knowledge that the set of defective complexes has cardinality at

Table 7.2: Outcome sets for defined scenarios.

route label	route	$\boldsymbol{a}$	$\boldsymbol{a}'$	$\boldsymbol{a}''$	$\boldsymbol{a}'''$
$\boldsymbol{r}_1$	1 1 0 0	1	1	1	1
$\boldsymbol{r}_2$	1 0 1 0	1	1	0	0
$\boldsymbol{r}_3$	1 0 0 1	1	1	0	0
$\boldsymbol{r}_4$	0 1 1 0	1	1	1	0
$\boldsymbol{r}_5$	0 1 0 1	1	0	0	0
$\boldsymbol{r}_6$	0 0 1 1	0	0	0	1

most 2 and rank at most 2, we now show how in each case the design can be used to identify the set of defective complexes using Algorithm 7.44.

First observe that, since $e = 2$, we have

$$\begin{aligned}\mathcal{E} \;=\; & \{\{\boldsymbol{c}_1\},\{\boldsymbol{c}_2\},\{\boldsymbol{c}_3\},\{\boldsymbol{c}_4\},\\ & \{\boldsymbol{c}_1,\boldsymbol{c}_2\},\{\boldsymbol{c}_1,\boldsymbol{c}_3\},\{\boldsymbol{c}_1,\boldsymbol{c}_4\},\{\boldsymbol{c}_2,\boldsymbol{c}_3\},\{\boldsymbol{c}_2,\boldsymbol{c}_4\},\{\boldsymbol{c}_3,\boldsymbol{c}_4\}\}\end{aligned}$$

- In the first scenario, it is trivial to show that the only elements of $\mathcal{E}$ whose intersection is not included in the outcome vector $\boldsymbol{a}$ are $\{\boldsymbol{c}_3\}$, $\{\boldsymbol{c}_4\}$ and $\{\boldsymbol{c}_3, \boldsymbol{c}_4\}$. That is, $\mathcal{G}$ contains all supersets of either $\boldsymbol{c}_1$ or $\boldsymbol{c}_2$. It follows immediately that $\mathcal{D} = \{\{\boldsymbol{c}_1\},\{\boldsymbol{c}_2\}\}$ is the set of defective complexes.
- In the second scenario, $\mathcal{G}$ contains $\{\boldsymbol{c}_1\}$, $\{\boldsymbol{c}_1, \boldsymbol{c}_2\}$, $\{\boldsymbol{c}_1, \boldsymbol{c}_3\}$, $\{\boldsymbol{c}_1, \boldsymbol{c}_4\}$, and $\{\boldsymbol{c}_2, \boldsymbol{c}_3\}$. After applying Step 3 we obtain $\mathcal{D}' = \{\{\boldsymbol{c}_1\},\{\boldsymbol{c}_2, \boldsymbol{c}_3\}\}$.
- In the last two scenarios, the set of defective complexes has a uniform rank of two, and the sets $\mathcal{D}''$ and $\mathcal{D}'''$ are obtained after applying only the first two steps of Algorithm 7.44.

It is clear that Step 2 of Algorithm 7.44 has complexity linear in $|\mathcal{E}| = \sum_{\varepsilon=1}^{e} \binom{n}{\varepsilon}$, where n is the number of hosts. As a result we observe that a set of defective complexes of cardinality at most d and rank e can be readily identified if a (d, e)-disjunct route design is used (as long as e and n are of bounded size).

7.3.3 Some GTC constructions

Designs that satisfy Definitions 7.26 and 7.39 have been studied by a variety of authors [48, 63, 81, 112, 120, 181, 190]. We next give two constructions from the literature that are of potential use in our setting.

Proposition 7.46. (D'yachkov et al. [47]) *Let d, e, t and w be positive integers such that $e \leq w \leq t - d$, and suppose M is the $\binom{t}{w} \times t$ binary matrix whose rows consist of all possible binary vectors of length t and weight w. Then M is (d, e^*)-disjunct.*

Example 7.47. The route design given in Table 7.1 is an example of Proposition 7.46 for the case $d = e = 2$, $t = d + e = 4$ and $w = 2$. Hence, this matrix is $(2, 2^*)$-disjunct, as discussed in Example 7.38.

Theorem 7.48. (Mitchell and Piper [120]) *An incidence matrix of a $(d+e)$-design, where the blocks correspond to rows and the elements to columns, is a (d, e)-KDP (and, by Theorem 7.40, is (d, e)-disjunct).*

Theorem 7.48 implies that we can use a $(d+e)$-(v, b, r, k, λ) design to find up to d defective complexes of size at most e within a set v of hosts, by sending b spy agents, where each host meets r spy agents and each agent visits k hosts. However, given that $b \geq vt/2$ for a t-design (if $v > k/2$) [15], t-designs for $t > 3$ are unattractive for use in GTC since the number of tests required is considerably greater than the number of test items. However, for spy agent applications, we are primarily interested in maximising k (i.e. the spy agent route length, see also §3.4.3.2). Hence, t-designs have some value for spy agent testing. We discuss this further in the examples provided in §7.5.4.

A (d, e)-disjunct route design can be used to identify complex defectives;

however, we have not discussed how individual malicious hosts can be identified, given a set of defective complexes. This is the subject of the next section.

7.4 Identifying individual malicious hosts

7.4.1 Problem representation

The objective of spy agent testing, in this chapter, is to identify individual malicious hosts (potentially of multiple `Types`) by testing (large) groups of hosts. We first consider how the characteristics of a set of malicious hosts define the set of defective complexes.

We start by introducing some further notation. Let $P(X, k)$ denote the set of all k-subsets of the set X. Clearly, $|P(X, k)| = \binom{|X|}{k}$, $|X| \geq k$. Formally we let $P(X, k) = \emptyset$, if $|X| < k$.

We first observe the following trivial result.

Lemma 7.49. *If all malicious hosts have* `Type` e *(for some* $e > 0$*), then the set of defective complexes is* $P(\Delta, e)$*, where* Δ *is the set of malicious hosts.*

Corollary 7.50. *If all malicious hosts have* `Type` e *(for some* $e > 0$*), then the set of defective complexes is non-empty if and only if* $|\Delta| \geq e$*, where* Δ *is the set of malicious hosts.*

Example 7.51. Suppose that there is only one defective host and it is of `Type-2`. Then, by Corollary 7.50, the set of defective complexes is empty. That is, a `Type-2` host will only misbehave if there is at least one other malicious host present.

The following algorithm gives a means of computing the set of defective complexes given a particular set of malicious hosts. The reverse process, i.e.

identifying the malicious hosts given a set of defective complexes, is discussed in §7.4.2.

Algorithm 7.52. *Suppose that the set Δ of malicious hosts is partitioned into the sets $\boldsymbol{\delta}_\varepsilon$ of the* `Type-`ε *(malicious) hosts, $1 \leq \varepsilon \leq e$ (where e is the largest occurring* `Type` *of a malicious host). The set of defective complexes can be constructed using the following procedure.*

- **Step 1***: Let*
$$D^* = \bigcup_{\varepsilon=1}^{e} P(\boldsymbol{\delta}_\varepsilon, \varepsilon)\,. \tag{7.4.1}$$
- **Step 2** *(which applies only if $e > 2$): For every ε ($2 \leq \varepsilon < e$), let $\Delta_{\varepsilon+1} = \bigcup_{i=\varepsilon+1}^{e} \boldsymbol{\delta}_i$. Then put*
$$D^{\{\varepsilon\}} = \bigcup_{k=1}^{\varepsilon-1} \{X \cup Y \ : \ X \subseteq \boldsymbol{\delta}_\varepsilon, |X| = k, Y \subseteq \Delta_{\varepsilon+1}, |Y| = \varepsilon - k\}\,. \tag{7.4.2}$$
- **Step 3***: The set of defective complexes is*
$$\mathcal{D} = D^* \cup \left(\bigcup_{\varepsilon=2}^{e-1} D^{\{\varepsilon\}} \right). \tag{7.4.3}$$

Proof of correctness. Since (from Lemma 7.17) the set of defective complexes is unique it is sufficient to show that $\mathcal{D}$ satisfies the two properties of Definition 7.13. To establish the first property we need to show that a route will give a positive result if and only if it contains an element of $\mathcal{D}$. Suppose $D_i \in \mathcal{D}$ and $|D_i| = \varepsilon$. Then, by definition, D_i will contain at least one `Type-`ε' malicious host ($2 \leq \varepsilon' \leq \varepsilon$) and $\varepsilon - 1$ other malicious hosts. Hence any route containing D_i as a subset will yield a positive result. Now consider a route that gives a positive result. From Definition 7.1, this route must contain a set A containing a `Type-`ε host, $\boldsymbol{c}_j$ and $\varepsilon - 1$ other malicious hosts, for some $\varepsilon \geq 1$. Suppose that $\boldsymbol{c}_a \in A$ is a `Type-`a host, and that this is the malicious host with

the smallest Type in A, $a \leq \varepsilon$. Take any a-subset of A containing $\boldsymbol{c}_a$. This subset is contained in $\mathcal{D}$ since if all hosts in this subset have the same Type then it is contained in $P(\boldsymbol{\delta}_a, a)$, and otherwise, it is contained in $D^{\{a\}}$. To prove the second property it is sufficient to show that all the pairs of elements in $\mathcal{D}$ are mutually non-inclusive, which follows immediately by definition. □

Lemma 7.53. *If $D^{\{\varepsilon\}}$ is as defined in Algorithm 7.52, then:*

$$|D^{\{\varepsilon\}}| = \sum_{k=1}^{\varepsilon-1} \binom{|\boldsymbol{\delta}_\varepsilon|}{k} \binom{|\Delta_{\varepsilon+1}|}{\varepsilon - k}. \tag{7.4.4}$$

Proof. As defined, the number of complexes in $D^{\{\varepsilon\}}$ is equal to the number of different ways of choosing the first k elements of an ε-complex from the set $\boldsymbol{\delta}_\varepsilon$ of Type-ε hosts ($1 \leq k \leq \varepsilon - 1$), multiplied by the number of all different ways of choosing the last $\varepsilon - k$ elements of this ε-complex from the set $\Delta_{\varepsilon+1}$ of Type-$(\varepsilon + 1)^+$ hosts. □

Example 7.54. Consider a set of hosts $\{\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_n}\}$ where $\mathbf{c_1}$ is of Type-1, $\mathbf{c_2}$ is of Type-2, and the other hosts are non-defective. From Algorithm 7.52 we have $\mathcal{D} = D^* = \{\{\mathbf{c_1}\}\}$. Since there is no defective complex containing $\mathbf{c_2}$, there is no way for the operator of a spy agent system to know whether or not $\mathbf{c_2}$ (or, indeed, any of the hosts other than $\mathbf{c_1}$) is malicious of Type greater than 1. More generally, the same problem will arise for all malicious hosts of Type $e > 1$ if there are less than e malicious hosts of Type greater than 1 present.

Example 7.55. Consider a set of hosts $\{\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_n}\}$, where $\mathbf{c_1}$ is of Type-1, $\mathbf{c_2}$ and $\mathbf{c_3}$ are of Type-2, and the other hosts are non-defective. From Algorithm 7.52 we have $\mathcal{D} = D^* = \{\{\mathbf{c_1}\}, \{\mathbf{c_2}, \mathbf{c_3}\}\}$. In this case the spy agent operator can successfully deduce that $\mathbf{c_1}$ is a Type 1 defective and that at least one of $\mathbf{c_2}$ and $\mathbf{c_3}$ is a Type 2 defective.

Example 7.56. Consider a set of hosts $\{\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_n}\}$ where $\mathbf{c_1}$ is of `Type-1`, $\mathbf{c_2}$ and $\mathbf{c_3}$ are of `Type-2`, $\mathbf{c_4}$ is of `Type-`x, $x > 2$, and the other hosts are non-defective. From Algorithm 7.52 we have:

$$\begin{aligned} D^* &= \{\{\mathbf{c_1}\}, \{\mathbf{c_2}, \mathbf{c_3}\}\} \\ \Delta_3 &= \{\mathbf{c_4}\} \\ D^{\{2\}} &= \{\{\mathbf{c_2}, \mathbf{c_4}\}, \{\mathbf{c_3}, \mathbf{c_4}\}\} \\ \mathcal{D} &= D^* \cup D^{\{2\}} \\ &= \{\{\mathbf{c_1}\}, \{\mathbf{c_2}, \mathbf{c_3}\}, \{\mathbf{c_2}, \mathbf{c_4}\}, \{\mathbf{c_3}, \mathbf{c_4}\}\}. \end{aligned}$$

We observe that, in this example, the spy agent system cannot distinguish between the case where $\mathbf{c_2}$, $\mathbf{c_3}$ and $\mathbf{c_4}$ are all of `Type` 2, and the case where either or both of them are of `Type` 2 and the other (or others) have `Type` greater than 2. Through this example, we can also observe that a malicious host ($\mathbf{c_4}$, in this example) might still be included in defective complexes (and be identified) even if it never misbehaves. In this example this arises when $x > 2$ ($\mathbf{c_4}$ is of `Type` 3^+). In this case the fact that $\mathbf{c_4}$ is malicious is revealed by other malicious hosts ($\mathbf{c_2}$ and $\mathbf{c_3}$) which will generate a positive result if and only if they are aware of the appearance of another malicious host in the route.

The above examples motivate the following lemma.

Lemma 7.57. *Suppose that the set Δ of malicious hosts contained in a particular set of hosts is partitioned into the sets $\boldsymbol{\delta}_\varepsilon$ of `Type-`ε (malicious) hosts. If there are hosts of `Type` greater than 1, let $\mu > 1$ be the `Type` of the malicious host with the smallest `Type` amongst the hosts in $\Delta - \boldsymbol{\delta}_1$; otherwise put $\mu = 1$. Then it follows that all malicious hosts are included in at least one defective complex if and only if $|\cup_{i \geq \mu} \boldsymbol{\delta}_i| \geq \mu$.*

Proof. We assume that Δ is non-empty throughout (otherwise the lemma holds trivially). First suppose that $\mu = 1$, i.e. $\Delta = \boldsymbol{\delta}_1$. In this case each defective host will be contained in exactly one defective complex. Also, we must clearly have $|\cup_{i\geq\mu} \boldsymbol{\delta}_i| = |\boldsymbol{\delta}_\mu| \geq 1$

Now suppose that $\mu > 1$. We consider two sub-cases. If there are no hosts of `Type` greater than μ, then (using the notation of Algorithm 7.52) $D^{\{\mu\}}$ will be empty and $|\cup_{i\geq\mu} \boldsymbol{\delta}_i| \geq \mu$ if and only if $|P(\boldsymbol{\delta}_\mu, \mu)| > 0$. However, from Algorithm 7.52 we know that a host of `Type` μ will be included in a defective complex if and only if $P(\boldsymbol{\delta}_\mu, \mu)$ is non-empty. The result follows in this sub-case.

Finally suppose $\mu > 1$ and that there are hosts of `Type` greater than μ. It follows that (using the notation of Algorithm 7.52) $|\cup_{i\geq\mu} \boldsymbol{\delta}_i| \geq \mu$ if and only if $|D^{\{\mu\}}| > 0$. If $D^{\{\mu\}}$ is non-empty then, by definition, every defective host of `Type` greater than or equal to μ will be included in an element of $D^{\{\mu\}}$. Equally, if any defective host of `Type` greater than μ is included in a defective complex then $D^{\{\mu\}}$ must be non-empty, and the result follows. □

Finally we comment on the rationale of representing the spy agent problem as a GTC problem. Consider Example 7.55, in which the malicious hosts, $\boldsymbol{c}_1$, $\boldsymbol{c}_2$ and $\boldsymbol{c}_3$, can be trivially identified by sending one spy agent to each host and one to each pair of hosts. However, this trivial solution is not attractive in our context as we are interested in maximising the route length. Also the identification of some or all defectives becomes harder when the number of `Type-1` and `Type-2` defective complexes is unknown. In the most general case, the trivial solution would require sending a spy agent to each subset of the set of all hosts. By using GTC designs we aim to: a) reduce the number of tests, and b) increase the route length.

7.4.2 Host classification

Algorithm 7.52 provides us with a means of identifying the set of defective complexes given a particular set of malicious hosts. However, as discussed in the examples in §7.4.1, this mapping is not injective; as a result, identifying the malicious hosts (and their `Types`) given a set of defective complexes is not always possible. In this section we consider this host classification problem in greater detail.

We first consider the special case where all the malicious hosts have the same `Type`; in this case we have the following simple result, which follows immediately from Lemma 7.49.

Lemma 7.58. *Suppose the set Δ of malicious hosts contains only `Type`-e hosts and $|\Delta| \geq e$. Then $\Delta = \bigcup_{D_i \in \mathcal{D}} D_i$, where $\mathcal{D}$ is the set of defective complexes.*

The following lemma follows trivially.

Lemma 7.59. *If a route design is $(\binom{d}{e}, e^*)$-disjunct and $e \leq d$, then it is a $(\overline{d}, e)$-classifier.*

We note that if all hosts have the same `Type` then the set of defective complexes has uniform rank (equal to the `Type`). However, the opposite is not always true. For example, consider a modified version of Example 7.56 in which $\boldsymbol{c}_1$ is a non-malicious host and $\mathcal{D} = \{\{\mathbf{c_2}, \mathbf{c_3}\}, \{\mathbf{c_2}, \mathbf{c_4}\}, \{\mathbf{c_3}, \mathbf{c_4}\}\}$. As discussed above, and following the analysis in Algorithm 7.52, the same set $\mathcal{D}$ would be obtained if all three of $\mathbf{c_2}$, $\mathbf{c_3}$ and $\mathbf{c_4}$ were of `Type` 2, and if two were of `Type` 2 and the other of larger `Type`.

In the case where there are mixed Types of malicious hosts, there is generally no unique host classification. The following two lemmas follow immediately from Algorithm 7.52.

Lemma 7.60. *If $\mathcal{D}$ is a set of defective complexes, then all the hosts in the set $\{\mathbf{c}_i | \mathbf{c}_i \in D_j, D_j \in \mathcal{D}\}$ are defective hosts.*

Lemma 7.61. *If D_i is a defective complex then all the elements of D_i are hosts of Type e, for some $e \geq |D_i|$.*

Using Lemmas 7.57 and 7.61 we can classify all malicious hosts as the following corollary suggests.

Corollary 7.62. *Suppose a set of hosts contains a set Δ of malicious hosts and the elements of the corresponding set $\mathcal{D}$ of defective complexes jointly contain all the malicious hosts. If a route design is $(|\mathcal{D}|, r)$-disjunct (or $(|\mathcal{D}|, r)$-separable), then the route design is a $(|\Delta|, \overline{e})$-classifier, where r is the rank of $\mathcal{D}$ and e is the largest Type of a host of Δ.*

7.5 A rank-two Type classification algorithm

7.5.1 Standard classification scenario

In the standard rank-e Type classification problem we consider malicious hosts with Type at most e, defective complexes with cardinality at most e, and a set of defective complexes with rank at most e. The hosts can be classified with the use of a GTC route design as long as the conditions established in Lemma 7.57 are satisfied.

We focus on the simple case where the set Δ of defectives contains a mixture of Type-1 and Type-2 defectives. As discussed in Example 7.54, if there is only one Type-2 defective, no route design can determine whether or not this

host is malicious, as this host is not included in the set of defective complexes. This observation follows immediately from Lemma 7.57, where in this case the necessary condition is: $|\cup_{i\geq 2}\boldsymbol{\delta}_i| = |\boldsymbol{\delta}_2| \geq 2$.

Applying Algorithm 7.52 in this special case, we obtain:

$$\mathcal{D} = D^* = \boldsymbol{\delta}_1 \cup P(\boldsymbol{\delta}_2, 2)\,. \tag{7.5.1}$$

Observe that, in this scenario, the set of malicious hosts and their corresponding `Types` can be trivially identified given the (identified) set of defective complexes.

7.5.2 Classification scenario with design restrictions

In this section we consider a slightly modified rank-2 `Type` classification problem. We modify Definition 7.1 to the following (the changes are underlined).

Definition 7.63. *Suppose $e \geq 1$. When processing a mobile agent whose route contains $d \geq 0$ malicious hosts, a `Type-`e (malicious) host will behave as follows:*

- *if $d \geq e$ and the route length is at least two then the host will handle the agent in such a way that it returns a positive result; and*
- *if $d < e$ then the host will handle the agent in such a way that a negative result will ensue unless another host abuses the agent.*

The rationale for the above modification is that it seems reasonable to assume that `Type-1` hosts will not misbehave if they are the only hosts included in the spy agent route. This restriction is consistent with the path length assumption discussed in §3.4.3.2.

The modified problem can be addressed in two different ways. Firstly, we could simply choose route designs that are classifiers and that satisfy the

route length restriction. An example of a suitable design has been studied in Example 7.38. Secondly, the set of defective complexes could be constructed in accordance with the modified host behavioural model. In this section we discuss the second approach (and its rationale).

To illustrate how the modified problem changes the situation, suppose that a route $\mathbf{r_1}$ only contains one host, $\mathbf{c_1}$, which happens to be `Type-1`. In the standard classification problem, the complex $\{\mathbf{c_1}\}$ is a defective complex. In our modified classification problem, $\{\mathbf{c_1}\}$ is no longer a defective complex because the route $\mathbf{r_1}$ will give a negative result.

Following from the above example, it is straightforward to show that the set of defective complexes $\mathcal{D}$ is made up of the 2-subsets of hosts that include at least one `Type-1` host and the 2-subsets of `Type-2` hosts. That is:

$$\mathcal{D} = P(\boldsymbol{\delta}_2, 2) \cup (P(S, 2) - P(S - \boldsymbol{\delta}_1, 2)) \ , \tag{7.5.2}$$

where S is the set of all hosts.

From (7.5.2) (observing that $P(\boldsymbol{\delta}_2, 2)$ is disjoint from $P(S, 2) - P(S - \boldsymbol{\delta}_1, 2)$), we obtain:

$$|\mathcal{D}| = \binom{|\boldsymbol{\delta}_2|}{2} + \binom{n}{2} - \binom{n - |\boldsymbol{\delta}_1|}{2} \ . \tag{7.5.3}$$

In this new problem, the modified host behaviour model leads to the construction of a different set of defective complexes, which has a uniform rank. Hence, to identify $\mathcal{D}$ it sufficient to use a $(|\mathcal{D}|, 2^*)$-disjunct or a $(|\mathcal{D}|, 2^*)$-separable route design.

One benefit of this new approach is that (d, e^*)-disjunct designs have been studied more extensively than other classes of disjunct design and more constructions are known—see, for example, [112].

However, this approach clearly increases the size $|\mathcal{D}|$ of the set of defective

complexes, if there are Type-1 hosts. Also, it complicates the process of identifying the individual defective hosts given a set of defective complexes, as Lemma 7.60 no longer applies. In particular, a defective complex may now contain a non-defective host. This problem is studied in the next section, where we provide a naive classification algorithm.

7.5.3 The algorithm

Consider a set of n hosts containing a set of d defective hosts ($0 \leq d < n-1$) that are of Type-1 or Type-2. Suppose also that a malicious host will only misbehave if the route length is at least two.

The classification algorithm described below requires the use of a $(|\mathcal{D}|, 2^*)$-disjunct route design, where $|\mathcal{D}|$ is as in (7.5.3). This route design is used to determine the set of defective 2-complexes, using Algorithm 7.42.

Algorithm 7.64 (Resilient rank-2 classification)**.** *Suppose a set of hosts S contains a set Δ malicious hosts of Type at most 2 and that a malicious host will only mishandle an agent if the agent visits at least 2 hosts. The following two-part algorithm partly or completely identifies Δ (depending on the number of Type-2 defective hosts). Note that two alternatives are provided for the second part of the algorithm.*

A. Identify Type-1 hosts.

Require: *A $(|\mathcal{D}|, 2^*)$-disjunct route design, and $|\Delta| < n-1$.*

Using Algorithm 7.42, identify the set $\mathcal{D}$ of defective complexes (which, from (7.5.2), has uniform rank 2).

for $i = 1$ *to* $i = n$ ***do***

Construct the set $\boldsymbol{T}_i$ *of the 2-complexes that contain* $\boldsymbol{c}_i$*:*

$\boldsymbol{T}_i = \{\{\boldsymbol{c}_i, \boldsymbol{c}_j\} \,|\, j \neq i\}$, $|\boldsymbol{T}_i| = n - 1$.

if $\boldsymbol{T}_i \subseteq \mathcal{D}$ ***then***

$\boldsymbol{c}_i$ *is* `Type`-1 *(otherwise* $|\Delta| \geq n - 1$*).*

Add $\boldsymbol{c}_i$ *to the set* $\boldsymbol{\delta}_1$.

end if

end for

B. Identify `Type`-2 *hosts.*

Require: *Part A (above).*

Put: $B = \mathcal{D} - (P(S, 2) - P(S - \boldsymbol{\delta}_1, 2))$.

if $|\boldsymbol{B}| \neq 0$ ***then***

$\boldsymbol{c}_i \in \cup B$ *are* `Type`-2*, i.e.* $\boldsymbol{\delta}_2 = \{\boldsymbol{c}_i \,|\, \boldsymbol{c}_i \in B_j, B_j \in B\}$.

$\boldsymbol{c}_i \notin \boldsymbol{\delta}_1 \cup \boldsymbol{\delta}_2$ *are non-defective.*

else

Within the set $S - \boldsymbol{\delta}_1$ *there are at least* $(n - |\boldsymbol{\delta}_1| - 1)$ *non-defective hosts and at most one* `Type`-2 *host, which cannot be identified.*

end if

B-alt. Identify `Type`-2 *hosts (alternative).*

Require: *Part A (above).*

for *All* $\boldsymbol{c}_i$ *in* $S - \boldsymbol{\delta}_1$ ***do***

for *All* $D_j \in \mathcal{D}$ ***do***

if $\boldsymbol{c}_i \in D_j$ *and* $D_j \cap \boldsymbol{\delta}_1 = \emptyset$ ***then***

Add $\boldsymbol{c}_i$ to the set $\boldsymbol{\delta}_2$.

Break inner loop

end if

end for

end for

if $|\boldsymbol{\delta}_2| = 0$ ***then***

There is at most one `Type-2` *host and at least* $n - |\boldsymbol{\delta}_1| - 1$ *non-defective hosts.*

end if

Ensure: $\boldsymbol{\delta}_1 \cap \boldsymbol{\delta}_2 = \emptyset$, $|\boldsymbol{\delta}_1| + |\boldsymbol{\delta}_2| < n - 1$, *and (7.5.3) hold.*

End of algorithm.

Proof of correctness. In part A, the algorithm identifies the `Type-1` hosts, given the set $\mathcal{D}$ of positive 2-complexes obtained by Algorithm 7.42. We can determine whether or not a host $\boldsymbol{c}_i$ is a `Type-1` defective as follows.

- If $\mathcal{D}$ contains all $n-1$ possible 2-complexes containing $\boldsymbol{c}_i$, then $\boldsymbol{c}_i$ is a `Type-1` host. This follows since, if $\boldsymbol{c}_i$ was not a `Type-1` defective then all the $n-1$ other hosts would have to be defective, which contradicts the assumption that $|\Delta| < n - 1$.

In part B, the algorithm attempts to identify the `Type-2` hosts and the remaining non-defectives. To do so, we consider the a subset B of the set $\mathcal{D}$ of defective complexes consisting of the defective complexes that do not contain any of the `Type-1` hosts identified in part A. We consider the following subcases.

- If B is non-empty then the hosts contained in all the 2-complexes in B must be Type-2 hosts (and there are at least two such hosts).
- Otherwise, there is at most one Type-2 host. It cannot be determined whether or not there is a single Type-2 defective host.

In the alternative version of part B, we consider the hosts that appear in a defective complex and that are not Type-1 hosts. If such a host appears in at least one defective complex which does not contain a Type-1 host, then both elements of this 2-complex must be Type-2 hosts.

Finally, the 'Ensure' step is present to check that the results given by the algorithm are consistent; inconsistencies may arise if the assumptions on which the algorithm depends regarding the number of types of malicious hosts do not hold. □

The following observations are key to the correct operation of Algorithm 7.64.

Remark 7.65. *Suppose a set of n hosts contains less than $n-1$ malicious hosts, each of which are of Type at most 2. If all the 2-complexes containing a host $\boldsymbol{c}_j$ are defective complexes, then $\boldsymbol{c}_j$ is a Type-1 host.*

Remark 7.66. *Suppose a set of n hosts contains only Type-1 or Type-2 malicious hosts. If at least one (but not all) of the 2-complexes containing a host $\boldsymbol{c}_j$ are defective complexes, and at least one of these 2-complexes contains another host $\boldsymbol{c}_k$ that is not a Type-1 defective, then $\boldsymbol{c}_j$ is a Type-2 defective (and so is $\boldsymbol{c}_k$). If some (but not all) of the 2-complexes containing a host $\boldsymbol{c}_j$ are defective complexes, and all these 2-complexes contain another host that is a Type-1 defective, then $\boldsymbol{c}_j$ is either non-defective or is the only Type-2 defective.*

7.5.4 Example implementation

We conclude this discussion of the rank 2 case by describing a series of experiments performed to demonstrate the operation of Algorithm 7.64.

Using Theorem 7.48 we can construct a $(2, 2)$-disjunct route design from a 4-design as follows. Consider the 4-$(11, 5, 1)$ design derived from the Mathieu Group M_{11} [38], which has $b = 66$ blocks, and $r = 30$. Associating blocks with rows (routes) and the elements to columns (hosts) we obtain a route design with 66 routes, 5 hosts per route, that can test 11 hosts, and in which each host is visited by 30 agents.

The 66 blocks of the 4-$(11, 5, 1)$ design are listed in Table 7.3. The sets of blocks incident with each of the 11 points (i.e. the columns of the spy agent route design) are given in Table 7.4.

Using a computer program (see Appendix A), we checked that, as expected, the above 4-design is $(5, 2)$-disjunct and a $(5, 2)$-KDP by checking that (7.3.2) and (7.3.3) hold for the case $d = 5$ and $e = 2$.

Software was also developed to implement the following procedure.

1. **Input**: choose a set of `Type`-1 hosts and a set of `Type`-2 hosts.
2. **Outcome**: calculate the route design outcome, for the chosen distribution of malicious hosts using the Mathieu design.
3. **Decoding**: identify the defective complexes using Algorithm 7.44.
4. **Classification**: use Algorithm 7.64 to verify the given input.

Table 7.5 summarises some results from this routine for different given inputs. These experiments showed that, as expected, the 4-$(11, 5, 1)$ design

Table 7.3: The blocks of a Mathieu 4-$(11, 5, 1)$ design.

blocks	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2	2	3	3
	3	3	3	3	4	4	4	5	5	6	7	8	4	4
	4	5	6	9	5	6	7	6	7	9	8	10	5	7
	10	8	7	11	9	8	11	11	10	10	9	11	6	9
blocks	15	16	17	18	19	20	21	22	23	24	25	26	27	28
	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	3	3	3	3	3	4	4	4	4	4	5	5	5
	4	5	5	6	6	7	5	5	6	6	8	6	6	8
	8	7	9	8	10	8	7	10	7	9	9	7	8	9
	11	11	10	9	11	10	8	11	10	11	10	9	10	11
blocks	29	30	31	32	33	34	35	36	37	38	39	40	41	42
	1	1	2	2	2	2	2	2	2	2	2	2	2	2
	6	7	3	3	3	3	3	3	3	3	4	4	4	4
	7	9	4	4	4	5	5	6	7	8	5	5	6	7
	8	10	5	6	7	6	7	8	10	9	6	8	10	9
	11	11	11	9	8	10	9	11	11	10	7	10	11	10
blocks	43	44	45	46	47	48	49	50	51	52	53	54	55	56
	2	2	2	2	2	2	3	3	3	3	3	3	3	3
	4	5	5	5	6	6	4	4	4	4	4	5	5	5
	8	6	7	9	7	7	5	5	6	6	9	6	6	8
	9	8	8	10	8	9	7	8	7	8	10	7	9	10
	11	9	11	11	10	11	10	9	11	10	11	8	11	11
blocks	57	58	59	60	61	62	63	64	65	66				
	3	3	4	4	4	4	4	5	5	6				
	6	7	5	5	5	6	7	6	7	8				
	7	8	6	6	7	7	8	7	8	9				
	9	9	8	9	9	8	10	10	9	10				
	10	11	11	10	11	9	11	11	10	11				

Table 7.4: Dual of a Mathieu 4-$(11, 5, 1)$ design.

points	1	2	3	4	5	6	7	8	9	10	11
	1	1	1	1	2	3	3	2	4	1	4
	2	2	2	5	5	6	7	6	5	9	7
	3	3	3	6	8	8	9	11	10	10	8
	4	4	4	7	9	10	11	12	11	12	12
	5	5	13	13	13	13	14	15	14	17	15
	6	6	14	14	16	18	16	18	17	19	16
	7	7	15	15	17	19	20	20	18	20	19
	8	8	16	21	21	23	21	21	24	22	22
	9	9	17	22	22	24	23	25	25	23	24
	10	10	18	23	26	26	26	27	26	25	28
	11	11	19	24	27	27	29	28	28	27	29
	12	12	20	25	28	29	30	29	30	30	30
	13	31	31	31	31	32	33	33	32	34	31
	14	32	32	32	34	34	35	36	35	37	36
	15	33	33	33	35	36	37	38	38	38	37
	16	34	34	39	39	39	39	40	42	40	41
	17	35	35	40	40	41	42	43	43	41	43
	18	36	36	41	44	44	45	44	44	42	45
	19	37	37	42	45	47	47	45	46	46	46
	20	38	38	43	46	48	48	47	48	47	48
	21	39	49	49	49	51	49	50	50	49	51
	22	40	50	50	50	52	51	52	53	52	53
	23	41	51	51	54	54	54	54	55	53	55
	24	42	52	52	55	55	57	56	57	56	56
	25	43	53	53	56	57	58	58	58	57	58
	26	44	54	59	59	59	61	59	60	60	59
	27	45	55	60	60	60	62	62	61	63	61
	28	46	56	61	61	62	63	63	62	64	63
	29	47	57	62	64	64	64	65	65	65	64
	30	48	58	63	65	66	65	66	66	66	66

Table 7.5: Evaluation results for resilient rank-2 classification algorithm.

Case	Input	Outcome ($\boldsymbol{a}$)	GTC alg. result ($\mathcal{D}_2$)	Alg. result
1	$\boldsymbol{\delta}_1 = \{1\}$ $\boldsymbol{\delta}_2 = \{2,3\}$	$\{1,2,3,4,5,6,7,8,$ $9,10,11,12,13,14,$ $15,16,17,18,19,20,$ $21,22,23,24,25,26,$ $27,28,29,30,31,32,$ $33,34,35,36,37,38\}$	$\{\{1,2\},\{1,3\},$ $\{1,4\},\{1,5\},$ $\{1,6\},\{1,7\},$ $\{1,8\},\{1,9\},$ $\{1,10\},\{1,11\},$ $\{2,3\}\}$	As Input
2	$\boldsymbol{\delta}_1 = \emptyset$ $\boldsymbol{\delta}_2 =$ $\{1,2,3,4\}$	$\{1,2,3,4,5,6,7,8,$ $9,10,11,12,13,14,$ $15,16,17,18,19,20,$ $21,22,23,24,25,31,$ $32,33,34,35,36,37,$ $38,39,40,41,42,43,$ $49,50,51,52,53\}$	$\{\{1,2\},\{1,3\},$ $\{1,4\},\{2,3\},$ $\{2,4\},\{3,4\}\}$	As Input
3	$\boldsymbol{\delta}_1 = \{5\}$ $\boldsymbol{\delta}_2 = \{8\}$	$\{2,5,8,9,13,16,17,$ $21,22,26,27,28,31,$ $34,35,39,40,44,45,$ $46,49,50,54,55,56,$ $59,60,61,64,65\}$	$\{\{1,5\},\{2,5\},$ $\{3,5\},\{4,5\},$ $\{5,6\},\{5,7\},$ $\{5,8\},\{5,9\},$ $\{5,10\},\{5,11\}\}$	$\boldsymbol{\delta}_1 = \{5\}$ $\boldsymbol{\delta}_2 =$ "at most one def."

will classify any 5 defective hosts of `Type` at most 2, except for the case where there is either one `Type-2` host or no `Type-2` host.

7.6 Conclusions

This chapter has extended the results of Chapter 6 by weakening the assumption that a malicious host will always violate a visiting spy agent. We have applied GTC theory to a complex spy agent behavioural model in which some defective will only yield a positive outcome in the presence of (or after collusion with) other defectives. In particular we have looked at scenarios in which:

- some malicious hosts will always mishandle a visiting spy agent, whereas
- other malicious hosts will only violate the spy agent if they identify a certain number of other malicious hosts in the agent's route.

We have shown that combinatorial route designs can still be used to identify malicious hosts that exhibit these more sophisticated types of behaviour when certain criteria are met, and we have introduced a classification algorithm. In this direction we have derived fundamental properties of GTC theory and we have proposed a rank-2 classification algorithm. It seems likely that this algorithm could be generalised for use in higher rank problems, although this is expected to yield a larger number of cases where complete host classifications cannot be achieved. Further, we note that computer based experiments indicate that some of the designs discussed here can also be successfully used for malicious host classification in higher rank cases. This more effective than expected operation may arise because of the symmetry of the defective complexes that arise when analysing hosts of a range of different `Types`. The study of further properties of route designs in the case where the set of defective complexes has other sets of specific properties is an interesting topic for future research.

The results of this chapter are important as they can be used to develop more sophisticated spy agent route designs with increased resilience to complex malicious host behaviour, as compared with those discussed in Chapter 6. These results also contribute to the general art of group testing for complexes (e.g. they may be adapted for use in DNA complex group testing applications) and could potentially become useful in future group testing applications. Such research directions are further discussed in Chapter 11.

"Simplicity is the ultimate sophistication."

Leonardo da Vinci

8

Optimal multi-stage spy agent group testing

Contents

8.1 Synopsis

In this chapter we take a different approach to that described in Chapters 6 and 7, in that we consider sequential group testing (SGT). We introduce an optimal SGT algorithm that can be used to construct spy agent routes.

Basing spy agent routing on SGT schemes differs from the NGT based approach developed in Chapters 6 and 7, in that the choice of which hosts to test depends on the outcome of the previous test. This means that a sequential (or multi-stage) spy agent routing algorithm is only likely to be preferable to a NGT construction if the outcome of a test can be acquired after a relatively short delay.

The objectives of spy agent group testing were introduced in §6.2, where two optimisation parameters were defined, namely spy agent security and group testing economy. Spy agent security is improved when the route length is increased, and economy is improved when the number of spy agents (tests) is reduced. The problem of optimising these two parameters is non-trivial for NGT schemes. In this chapter we give a formal definition of spy agent routing optimality, and give a simple SGT solution.

The remainder of the chapter is organised as follows. Section 8.2 introduces the sequential spy agent GT scenario and formulates the design problem, and §8.3 addresses optimality issues for spy agent routing in SGT schemes. Finally, a simple optimal multi-stage algorithm is described and analysed in §8.4.

Aspects of the work described in this chapter have been published, [96].

8.2 Introduction

8.2.1 Scenario

In this chapter we consider the use of adaptive, i.e. sequential, GT for constructing spy agent routes. This may be preferable in spy agent applications where test outcomes are available quickly enough to enable all the tests to be completed in a reasonable time. This scenario contrasts with that described in §6.3.2 and considered in Chapters 6 and 7.

The choice of the general approach to spy agent route design, and how it depends on the application scenario, is discussed further in Chapter 10.

8.2.2 The problem

As discussed in §2.6.2, a fundamental objective of a GT algorithm is to minimise the total number of tests required to find all the malicious hosts within a set of hosts. In our particular application of GT our primary objective is to select spy agent routes so as to maximise the number of hosts that each spy agent visits, i.e. the route lengths. This objective is discussed in various places in this thesis, including §3.4.3.2, §5.4.3, §6.3.1 and §7.2.1.

In a NGT scheme, all the routes are predetermined. A carefully chosen route design can thus be designed to contain sufficiently long routes (see, for example, the NGT constructions given in §6.4.2 and §7.3.3).

One major problem with using a standard SGT algorithm (see, for example, [45]) in a spy agent setting is that it is likely to involve the use of very short routes. This is because the *a priori* knowledge of a route outcome is likely to reduce the size of the initial GT space. This, in turn, is likely to reduce the length of the routes of spy agents deployed later in the procedure. Thus a standard SGT algorithm may not be usable in a spy agent application.

This observation is treated more formally in Lemma 8.8 below.

We next give two examples illustrating the possible use of SGT in a spy agent scenario.

Example 8.1. Let $\{\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4\}$ be a set of hosts known in advance to contain exactly one defective. The single defective can be identified with $\log_2 4 = 2$ tests (the minimum number possible), as follows. First, test the subset $\{\boldsymbol{c}_1, \boldsymbol{c}_2\}$: if this gives a positive result, then the defective is either $\boldsymbol{c}_1$ or $\boldsymbol{c}_2$; otherwise, the defective is either $\boldsymbol{c}_3$ or $\boldsymbol{c}_4$. In either case, the defective can be identified with a further single-host test. This is an efficient GT algorithm (in terms of minimising the number of tests) but it is inappropriate for spy agent applications as a result of the use of a single-target agent.

Example 8.2. Let $\{\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4\}$ be a set of hosts known in advance to contain exactly one defective. Suppose all 3-subsets of the four hosts are tested in turn until a negative result is obtained. The defective host is then the one not contained in the route giving this negative result. In this case, the route length is 3, and superficial analysis suggests that between one and four tests are required, with an expected number of 2.5 tests. However, more careful analysis reveals that a fourth test will never be required since, after three positive tests, the fact that the fourth test will yield a negative result can be inferred. That is, this algorithm will always require between one and three tests, with an expected number of 2.25 tests. Whilst this is not much more than the minimum for this small case, for larger sets of hosts the worst case of this algorithm will require $n-1$ tests, i.e. no better than the trivial algorithm in which the hosts are tested one by one.

In the sections that follow we consider possible approaches to route design for SGT that maximise the lengths of spy agent routes. We propose a simple

SGT algorithm suitable for use in a spy agent scenario, and prove that it is optimal in a sense we define below.

8.3 Optimal spy agent sequential group testing

8.3.1 Assumptions

We make the same assumptions as in §6.3.1. We also suppose that all malicious hosts are of `Type-1`, as discussed in §7.2.1, i.e. a spy agent visiting a malicious host will always yield a positive outcome. Moreover, we assume that the number of malicious hosts is unknown, i.e. we consider the $S(\overline{n}, n)$ GT problem.

We next consider a measure of optimality for a SGT algorithm when used in a spy agent scenario. We first give some terminology.

8.3.2 Definitions

During operation of a SGT scheme, the choice of subsequent tests depends on previous test outcomes, and a test outcome depends on the number and distribution of malicious hosts but not on the order in which they are visited. As a result, we can make the following observation.

Remark 8.3. *Let $S(\overline{d}, n)$ be a sample space consisting of a set of n hosts containing at most d defectives. Since there are at most d defectives, there are $\sum_{i=0}^{d} \binom{n}{i}$ ways in which the defectives could be distributed amongst the set of n hosts, and hence during the operation of a SGT algorithm designed to cope with this number of defective hosts, up to $\sum_{i=0}^{d} \binom{n}{i}$ different sequences of routes may be used, depending on whether individual tests give a positive or negative result.*

As discussed in §8.2.2, given a SGT algorithm we wish to identify the minimum length of a route generated by this algorithm for all possible sample spaces $S(\overline{d}, n)$. This requirement is captured by the definition below.

Definition 8.4. *Let S be a set of n hosts containing at most d malicious hosts, let j be an identifier for a particular distribution of at most d malicious hosts within S, and let $\mathcal{A}$ be a SGT algorithm that is capable of identifying up to d malicious hosts. Suppose also that $\mathcal{A}$, given a particular distribution of at most d malicious hosts (with label j), uses a sequence of routes $Q_j = (R_1, R_2, \ldots, R_k)$, where R_i is the ith set of hosts to be tested, $1 \leq i \leq k$. Then a route R_i in Q_j is said to be a* weak route *in $\mathcal{A}$ if $|R_i| \leq |R_{i'}|$, for every route $R_{i'}$ in $Q_{j'}$, for every distribution j'.*

Definition 8.4 enables us to define a notion of optimality for SGT algorithms when used in a spy agent setting, as follows.

Definition 8.5. *Let S be a set of n hosts. Suppose that a SGT algorithm $\mathcal{A}$ can identify all the malicious hosts in S, regardless of their number and distribution, and let $\rho_{\mathcal{A}}$ be the length of a weak route in $\mathcal{A}$. $\mathcal{A}$ is said to be an optimal sequential spy agent routing algorithm, if, given any other SGT algorithm $\mathcal{B}$ that can also identify all the malicious hosts, $\rho_{\mathcal{A}} \geq \rho_{\mathcal{B}}$, where $\rho_{\mathcal{B}}$ is the length of a weak route in $\mathcal{B}$.*

☞ In this chapter, we use the terms *optimal sequential group testing algorithm*, *optimal sequential routing algorithm*, and *optimal algorithm* interchangeably to mean an optimal sequential spy agent routing algorithm, as defined above.

Example 8.6. In Example 8.2 the length of a weak route is $\rho_{\mathcal{A}} = 3$. The

given algorithm is optimal since the only route containing all 4 hosts cannot help to identify the malicious host.

8.3.3 Optimality properties

In this subsection we derive some properties of optimal spy agent routing algorithms.

Theorem 8.7. *Suppose a set of n hosts is known in advance to contain at most d defectives. Then the length ρ of a weak route in an optimal sequential spy agent routing algorithm capable of detecting all the malicious hosts satisfies $\rho = n - d$.*

Proof. Consider the route design consisting of all subsets of hosts of size $n-d$. This clearly has the property that the length of a weak route is $n-d$. We show that this route design is capable of identifying all the defective hosts (given there at most d of them).

There are at least $n - d$ well-behaved hosts, and hence every well-behaved host will appear in at least one route containing only well-behaved hosts, i.e. a route giving a negative result. Thus the union of all routes giving negative results contains all well-behaved hosts. However, it clearly cannot contain any defective hosts, since the presence of a defective host will always cause a route to give a positive result. Thus the complement of this union is precisely the set of defective hosts, and the route design can identify up to d defectives. Hence $\rho \geq n - d$.

Now suppose that there are exactly d defective hosts. Any route of size greater than $n - d$ will thus always contain at least one defective host and hence will always give a positive result. As a result, if a route design is to be

capable of identifying the defective hosts it must contain at least one route of size $n-d$, i.e. $\rho \leq n-d$, and the result follows. □

Lemma 8.8. *Suppose a set of n hosts contains exactly d defectives and let $\mathcal{A}$ be a SGT algorithm that can identify all the defectives. Suppose that for all possible distributions of the d defectives algorithm $\mathcal{A}$ requires a maximum of $N_{\max}$ routes (tests). If $\mathcal{A}$ is optimal, then $N_{\max} \geq \binom{n}{d} - 1$.*

Proof. If $\mathcal{A}$ is optimal then, from Theorem 8.7, all routes have length no less than $\rho = n-d$. Any route that contains more than $n-d$ hosts will always give a positive result (because that route will surely contain at least one defective), and, thus, such a test is redundant (because it provides no new information). It follows that $N_{\max}$ is minimised if all the routes of $\mathcal{A}$ have length less or equal to $n-d$. Thus, for an optimal $\mathcal{A}$, $N_{\max}$ is minimised if all the routes have length $n-d$. To identify the defective hosts for all possible distributions it is necessary and sufficient to consider all but one of the routes containing all the possible $(n-d)$-subsets of hosts. The result follows. □

We note that Lemma 8.8 implies that $N_{\max} \geq n-1$ if $1 \leq d < n$. This condition is to some extent inconsistent with the main objective of GT theory, which is to design algorithms with the property that $N_{\max}$ is significantly less than n, as discussed in §2.6.2. This observation suggests that known GT algorithms are not suitable in the context of the optimality criterion defined above; for this reason, we propose in the next section a new optimal routing algorithm.

8.4 Multi-stage sub-group routing algorithm

8.4.1 Objectives

In this section we present a special type of SGT algorithm which we call a *multi-stage sub-group routing algorithm*, and show that it is optimal.

The objective of the multi-stage sub-group routing algorithm is to construct routes that can identify all malicious hosts within a group of hosts, regardless of how they are distributed, in such a way that the length of a weak route is maximised. That is, we wish to devise an optimal algorithm, as given by Definition 8.5.

8.4.2 The algorithm

The algorithm operates as follows. As previously, we suppose that there are n hosts and that we wish to identify up to d malicious hosts. We construct and test routes in the following series of stages:

Stage 0: Deploy a single agent with a route containing all n target hosts. If the outcome is negative, the algorithm terminates and all hosts are deemed to be well-behaved.

Stage 1: Deploy a set of n agents, with routes equal to every possible $(n-1)$-subset of the n hosts. If one route tests negative then the algorithm terminates, and the single agent not in this route is deemed to be the only defective host. (In all other cases all routes will test positive, there are at least two defective hosts, and the algorithm continues).

...

Stage i $(1 < i < n)$: Deploy a set of $\binom{n}{i}$ agents, with routes equal to every possible $(n-i)$-subset of the n hosts. If at least one route tests negative,

then the algorithm terminates, and the complement of the union of the routes giving a negative result are deemed to be the set of defective hosts. (If all routes test positive then there are at least $i + 1$ defective hosts, and the algorithm continues).

...

Stage n: If all the above stages complete without the algorithm terminating then all hosts are defective.

We can describe this algorithm in a more formal way as follows.

Algorithm 8.9 (Multi-stage sub-group routing)**.**

Require: *the set S of n hosts contains some unspecified number of* `Type-1` *defective hosts.*

for $i = 0$ ***to*** $i = n - 1$ ***do***

Deploy a set of $\binom{n}{i}$ agents, with routes equal to every possible $(n-i)$-subset of S.

if *at least one route gives a negative result* ***then***

The set of well-behaved hosts is equal to the union of the routes giving a negative result.

Stop algorithm.

end if

end for

All hosts are defective

We note that Algorithm 8.9 can be modified by always terminating at stage k, for some k satisfying $1 \leq k < n$, if it is required that a route length

should be at least $n - k$. If, however, such a modification is required, then it is possible that the set of defective hosts will not be identified.

8.4.3 Correctness

We now show that Algorithm 8.9 performs as claimed.

Theorem 8.10. *Suppose Algorithm 8.9 is applied to a set S of n hosts. Then the following properties hold:*

(P1) if Algorithm 8.9 does not terminate at stage i $(0 \leq i \leq n-1)$, then there are at least $i+1$ defective hosts;

(P2) if Algorithm 8.9 terminates at stage i $(0 \leq i \leq n-1)$, then there are precisely i defective hosts, and if R (which must be non-empty) is the set of routes giving a negative result, then the set of defective hosts is equal to $S - \bigcup_{r \in R} r$;

(P3) Algorithm 8.9 is an optimal routing algorithm, regardless of the number of defective hosts.

Proof. We address each listed result as follows.

1) In stage i, non-termination means that every route gives a positive result. If there were at most i defective hosts, then there would exist at least one $(n-i)$-subset of hosts (i.e. a route) excluding all the defective hosts, i.e. there would exist a route giving a negative result, contradicting our assumption of non-termination, and hence the result follows.

2) From (P1), if the algorithm reaches stage i then there are at least i defective hosts. If there were more than i defective hosts then the algorithm would not terminate, since every $(n-i)$-subset of S would contain at least

one defective host, and hence every route would give a positive result. This contradicts our assumption of termination and thus (P2) follows.

3) From (P1) and (P2), if S contains precisely d defective hosts, then the algorithm will terminate at stage d, and hence the smallest route used will have size $n - d$. The result follows from Theorem 8.7. □

8.4.4 Efficiency

Whilst the multi-stage sub-group routing algorithm described above is optimal under our definition, it is clearly not efficient in terms of the number of tests required to identify the defective hosts. Indeed, if there is more than one defective host then it is clearly less efficient in this sense than simply testing the hosts one by one. The precise performance of the algorithm is captured by the following results.

Lemma 8.11. *If Algorithm 8.9 is applied to a set of n hosts containing precisely d defectives ($d < n$), then the number of tests performed will be equal to:*

$$A(n, d) = \sum_{i=1}^{d} \binom{n}{i}. \tag{8.4.1}$$

Proof. From Theorem 8.10, the algorithm will terminate at stage d. As described above, at stage i ($0 \leq i \leq n-1$), Algorithm 8.9 will involve performing a total of $\binom{n}{n-i} = \binom{n}{i}$ tests. The result follows. □

Lemma 8.12. *If Algorithm 8.9 is applied to a set of n hosts containing precisely d defectives ($d < n$), then the number of agents processed by each host will be equal to:*

$$C(n, d) = \sum_{i=0}^{d} \binom{n-1}{i}. \tag{8.4.2}$$

Proof. From Theorem 8.10, the algorithm will terminate at stage d. As described above, at stage i ($0 \leq i \leq n-1$), Algorithm 8.9 will involve performing a total of $\binom{n}{i}$ tests involving every possible $(n-i)$-subset of hosts. For any particular host, a total of $\binom{n-1}{n-i-1} = \binom{n-1}{i}$ of these tests will involve a route that visits that host. The result follows. □

8.4.5 Discussion

In this chapter we have considered optimality in route designs from one perspective, namely maximising the route size. However, as documented in §8.4.4 and Lemma 8.8, this has the effect that the number of tests required and the number of agents processed by each host can become very large. As discussed in §5.4.1, if a host is visited by a large number of spy agents then the accuracy of the test results may be compromised. It is thus clear that optimality needs to be considered from a variety of perspectives, and a more complex definition of optimality would be helpful.

One alternative approach would be to use other system parameters to evaluate the credibility of the GT identification results. We return to this problem in Chapter 9, where we introduce a method to evaluate spy agent results obtained from either NGT or SGT schemes.

8.5 Conclusions

In this chapter we have studied the application of SGT algorithms to the spy agent routing problem. This work complements the spy agent NGT and GTC problems, studied in Chapters 6 and 7 respectively.

Unlike standard NGT constructions, standard SGT algorithms yield routing schemes that are not appropriate for spy agent scenarios. This is because

SGT algorithms typically involve the use of routes containing very few hosts. To address this problem, in this chapter the problem of spy agent routing optimisation is formulated, and a new type of SGT algorithm is proposed that is optimal under the definition given.

The defined optimality criterion and the challenge of non-trivial malicious host behaviour give rise to further challenges, including the following.

- What alternative optimality definitions can be given?
    a. How can other spy agent system parameters (see §5.4) influence spy agent security?
    b. How can a GT algorithm be both (near) optimal and efficient in terms of the number of tests and the number of agents visiting each host?
- How can malicious hosts be identified when they behave in more complex ways?
    a. What happens if `Type-`e hosts are assumed?
    b. What happens if malicious hosts behave inconsistently?

These issues are possible topics for future research. In the next chapter we relax some of the assumptions made in Chapters 6, 7 and 8 by considering how the reliability of group testing identification results might depend on a number of spy agent system parameters.

"Quality is not an act, it is a habit."

Aristotle

9 Evaluating spy agent results

Contents

9.1 Synopsis

In this chapter we introduce a methodology for the evaluation of the credibility of spy agent classification results. We assume that the behaviour of a malicious host towards a visiting spy agent depends on a) the total number of hosts this spy agent visits (as discussed in 5.4.3), and b) random choice.

The remainder of the chapter is organised as follows. Section 9.2 revisits the spy agent evaluation problem. Section 9.3 introduces the assumptions underlying the proposed model, presents the spy agent credibility evaluation model, and describes its application to the NGT and SGT schemes introduced in previous chapters. Section 9.4 analyses the impact of random malicious host behaviour models, and, finally, §9.5 concludes this chapter.

Aspects of the work described in this chapter have been published, [97].

9.2 Introduction

9.2.1 Previous discussions

The evaluation of remote hosts is the main objective of a spy agent system. The evaluation process involves a set of trust assessments that can yield estimates for a variety of security issues, as discussed in §3.4.4 (and as analysed further in Chapter 10).

In Chapters 6, 7 and 8, trust evaluation was discussed in the context of identifying the malicious hosts from amongst a set of hosts, using the results obtained from a set of spy agents with specified routes. In such schemes, the fundamental spy agent security requirement is to construct spy agent routes in a manner that ensures that malicious hosts exhibit characteristic behaviour (see §4.4.2). The problem of designing appropriate sets of spy agent routes

was considered for two main host behaviour models. More specifically,

1. in §6.3.1 and §8.3.1 it is assumed that a malicious host always yields a positive spy agent outcome if the spy agent route length is sufficiently large; whereas

2. in §7.2.1 it is additionally assumed that a malicious host will only exhibit behaviour leading to a positive outcome if a certain number of other malicious hosts appear in the spy agent's route.

Also, as discussed in §6.5, error-tolerant GT designs can be used to identify malicious hosts exhibiting inconsistent behaviour (at least to a certain extent).

These spy agent routing schemes can be improved in a number of ways, as discussed in §6.5, §7.6 and §8.5. This is not the objective of this chapter; instead we attempt here to evaluate the credibility of given host evaluation results, i.e. the level of trustworthiness of the spy agent results acquired using the given schemes.

9.2.2 Problem description

The trustworthiness of spy agent results depends on the validity of the underlying assumptions for the application scenario. In this chapter we relax some of the assumptions previously made, and consider the following cases:

- a malicious host may not behave maliciously if the route of a visiting agent is insufficiently long;
- a malicious host may not behave maliciously for random reasons.

Example 9.1. Consider the NGT spy agent route design given in Example 6.21 (based on the Fano Plane) where, given a set of test results, the

design can be used to identify at most two malicious hosts. Now suppose that the first malicious host misbehaves if and only if the route length is greater than three, and the second malicious host misbehaves if and only if the route length is greater that two. Clearly, since all routes have length equal to three, the first malicious host will not misbehave and the design can only be used to identify the second malicious host.

Example 9.2. Consider again the spy agent route design given in Example 6.21, and suppose that the spy agents are sent sequentially. Also suppose that the hosts $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are malicious and will misbehave if and only if the route length is greater that two and they have been visited by at most two spy agents. Although all routes have length equal to three, the routes $\boldsymbol{r}_3 = \{\boldsymbol{c}_1, \boldsymbol{c}_6, \boldsymbol{c}_7\}$ and $\boldsymbol{r}_5 = \{\boldsymbol{c}_2, \boldsymbol{c}_5, \boldsymbol{c}_7\}$ will test negative regardless of the fact that $\boldsymbol{r}_3$ contains $\boldsymbol{c}_1$ and $\boldsymbol{r}_5$ contains $\boldsymbol{c}_2$. This is because $\boldsymbol{c}_1 = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3\}$ and $\boldsymbol{c}_2 = \{\boldsymbol{r}_1, \boldsymbol{r}_4, \boldsymbol{r}_5\}$, which means that the agent with route $\boldsymbol{r}_3$ is the third agent to visit $\boldsymbol{c}_1$, and the agent with route $\boldsymbol{r}_5$ is the third agent to visit $\boldsymbol{c}_2$. The outcome will be $\boldsymbol{a} = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_4\}$ and, using Corollary 6.18, we cannot identify any defective. This is because, in this case, the union of all negative routes contains all hosts: $\boldsymbol{r}_3 \cup \boldsymbol{r}_5 \cup \boldsymbol{r}_6 \cup \boldsymbol{r}_7 = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4, \boldsymbol{c}_5, \boldsymbol{c}_6, \boldsymbol{c}_7\}$.

Example 9.2 shows that, if some malicious hosts misbehave in a selective way, then the route design can yield inconsistent results. That is, while some routes test positive, the malicious hosts responsible cannot be identified. This means that, in this case, it can be deduced that one or more hosts are malicious, but that they have not mistreated all the spy agents they have processed.

More generally, when presented with an inconsistent set of spy agent test results, the following options for addressing the issue could be considered.

1. The evaluator could attempt to assess the probability that the individual hosts are malicious.

2. The evaluator could attempt to assess the probability that the obtained results are believable (which is discussed in this chapter).

3. The evaluator could retest the hosts using longer routes (e.g. by introducing additional hosts) and re-evaluate the new results.

Typically, retesting should only be performed after a delay long enough to minimise the likelihood that a potentially malicious host will associate the two tests.

Example 9.3. Let $S = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4\}$ be a set of hosts containing two defectives $\boldsymbol{c}_1$ and $\boldsymbol{c}_4$. Suppose that $\boldsymbol{c}_1$ misbehaves if and only if the agent route length is at least two, and $\boldsymbol{c}_2$ misbehaves if and only if the agent route length is at least three. Now consider the application of the multi-stage subgroup routing algorithm described in §8.4. At stage zero, the agent containing $\{\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4\}$ will test positive, implying the presence of at least one defective. At stage one, all four of the agents visiting 3-subsets of the four hosts will test positive. This implies the presence of at least two defectives. Stage two involves the use of six agents, each visiting a distinct 2-subset of the four hosts. Of these, only the three containing $\boldsymbol{c}_1$ will give a positive result. Host $\boldsymbol{c}_1$ is the only host that can tentatively be identified as malicious and the algorithm ends.

Example 9.4. Now suppose that $\boldsymbol{c}_1$ is the only well-behaved host in the set of hosts $S = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4\}$, and that the hosts $\boldsymbol{c}_2$, $\boldsymbol{c}_3$ and $\boldsymbol{c}_4$ will always misbehave if an agent has route length at least three. Suppose also that, if the

route length is equal to two, the hosts $\boldsymbol{c}_2$, $\boldsymbol{c}_3$ and $\boldsymbol{c}_4$ will misbehave randomly. Then the application of the multi-stage sub-group routing algorithm described in §8.4 will reach stage two, in which six agents are despatched each visiting a distinct 2-subset of the four hosts. Then $\boldsymbol{c}_2$, $\boldsymbol{c}_3$ and $\boldsymbol{c}_4$ might, by chance, choose to misbehave when handling the agents $\{\boldsymbol{c}_1, \boldsymbol{c}_2\}$, $\{\boldsymbol{c}_1, \boldsymbol{c}_3\}$, and $\{\boldsymbol{c}_1, \boldsymbol{c}_4\}$, respectively, but not for any other agents. This will yield exactly the same result that is observed in Example 9.3, i.e. the algorithm will identify host $\boldsymbol{c}_1$ as the only malicious host. We note that malicious hosts could choose to exhibit this behaviour through a desire to 'frame' $\boldsymbol{c}_1$.

In Example 9.3 it is clear that there is a discrepancy in the results, since from stage 1 there are known to be at least two malicious hosts, but only one ($\boldsymbol{c}_1$) can be tentatively identified. Also, from Example 9.4 we observe that when malicious hosts behave inconsistently, the obtained result ($\boldsymbol{c}_1$ is malicious) may be completely wrong.

Inconsistent host behaviour will not necessarily be obvious from the spy agent results. That is, whilst in Examples 9.2, 9.3 and 9.4 the results make it clear that at least one host has misbehaved in a selective way, scenarios could arise in which no such evidence is available from the test results. This possibility is highlighted in the following example.

Example 9.5. Suppose that, as in Example 9.4, $S = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4\}$ is a set of hosts in which $\boldsymbol{c}_1$ is the only well-behaved host, and that the hosts $\boldsymbol{c}_2$, $\boldsymbol{c}_3$ and $\boldsymbol{c}_4$ always misbehave if a visited agent has route length at least three. Then the application of the multi-stage sub-group routing algorithm described in §8.4 will reach stage two. Now suppose that, at this stage, $\boldsymbol{c}_2$, $\boldsymbol{c}_3$ and $\boldsymbol{c}_4$ misbehave when handling the agents they receive except for the one with route $\{\boldsymbol{c}_1, \boldsymbol{c}_2\}$. The algorithm will then identify hosts $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ as well-behaved, hosts $\boldsymbol{c}_3$ and

$\boldsymbol{c}_4$ as malicious, and there is no indication of any inconsistency in the results.

More generally, there may be a variety of reasons why test results cannot be trusted. Such a situation could arise if malicious hosts do not behave according to the assumptions on which the routing scheme is based. For example, as in the examples above, inconsistent results could be obtained when a malicious host chooses whether or not to misbehave depending on the agent route length and/or randomly.

The presence of inconsistent (or erroneous) results means that we need to use probabilistic arguments to try to determine which hosts are (and are not) defective. However, this is not the purpose of the assessment scheme discussed in this chapter. Instead, in the next section, we propose a method of evaluating the credibility of an obtained set of spy agent outcomes. That is, we wish to evaluate the probability that all the hosts contained in a spy agent route are non-malicious hosts when this agent yields no signs of abuse.

We note that the proposed method is based on less restrictive assumptions about possible malicious host behaviour than have applied in previous chapters. We propose a means to analyse results obtained from the application of a spy agent scheme without requiring any additional spy agents to be deployed.

9.3 Credibility evaluation model

9.3.1 Assumptions

A spy agent GT scheme has the objective of classifying target hosts as either malicious or non-malicious. To assess the credibility of such classification results, we make the following assumptions about host behaviour and its interpretation by the operator of the spy agent system.

(A-1) A malicious host cannot identify whether or not a visiting agent is a spy agent. [That is, the spy agent dissemblance requirements are met, as discussed in §4.3.5.]

(A-2) A malicious host has two modes of operation: a malicious mode and a non-malicious mode. A malicious host is in the malicious mode of operation with a certain probability called the *Probability of Malicious Mode* (PMM).

(A-3) A host's PMM depends only on the internal characteristics of the host. That is, the mode of operation of a malicious host is independent of any external events, including the existence and behaviour of other malicious hosts.

(A-4) A malicious host estimates that it can avoid being identified as malicious, if it misbehaves, with a certain probability. We call this probability the *Probability of Identification Avoidance* (PIA).

(A-5) A host's PIA depends on the spy agent route length; we suppose that the PIA increases linearly with the length of the agent route. This assumption is in line with the assumption that the longer the agent route, the less a malign host is likely to suspect a spying scenario, as discussed in §3.4.3.2. We note that in deterministic host behaviour models, such as those developed in Chapters 6, 7 and 8, we assume that the route length is always sufficiently long and that PIA = 1.

(A-6) A malicious host processes a visiting spy agent maliciously if and only if a) the host is in the malicious mode of operation, and b) the host decides it can avoid identification. Building on (A-2) and (A-4) we

suppose that a malicious host processes a visiting spy agent maliciously with a certain probability, which we call the *Probability of Characteristic Behaviour* (PCB), where PCB = PMM × PIA.

(A-7) A positive spy agent outcome is always credible in the sense that it always indicates the existence of at least one malicious host in the spy agent route.

(A-8) A negative spy agent outcome is not credible, in the sense that it may contain malicious hosts that do not behave maliciously. We associate a value (between 0 and 1) with a spy agent route which is an estimate for the probability that this route contains a malicious host and gives a negative result. We call this value the *Probability of Uncharacteristic Result* (PUR).

(A-9) We assign a value (between 0 and 1) for a set of spy agent classification results, which depends on the degree to which the outcome set of a spy agent routing scheme is believable. We call this value the *Credibility of Classification* (CC).

9.3.2 The model

In line with (A-2) and (A-3), we assume that a malicious host is always in either a Malicious Mode (MM) or a Non-malicious Mode (NM), and that the transitions between MM and NM are probabilistic. I.e. a malicious node behaves according to a Markov Model, as shown in Figure 9.1.

Based on this behavioural model, we define PMM as follows.

Definition 9.6. *Suppose that a host* $\boldsymbol{c}_j$ *is malicious and that its malicious*

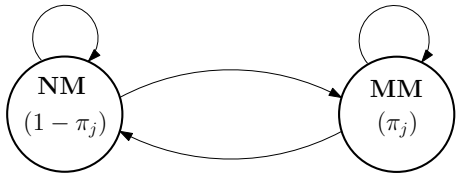


Figure 9.1: Markov model with two states: Non-malicious Mode (NM) and Malicious Mode (MM).

behaviour is characterised by a discrete random process with the Markov property. The PMM of $\boldsymbol{c}_j$ *(written* PMM_j*) is the steady-state probability that* $\boldsymbol{c}_j$ *is in the MM state:*

$$\mathrm{PMM}_j = \pi_j \,. \tag{9.3.1}$$

In line with (A-5), we define PIA as follows.

Definition 9.7. *Suppose that host* $\boldsymbol{c}_j$ *is malicious. The PIA of* $\boldsymbol{c}_j$ *w.r.t. a route* $\boldsymbol{r}_i$ *that contains* $\boldsymbol{c}_j$ *(written* PIA_{ij}*) is defined as follows:*

$$\mathrm{PIA}_{ij} = \begin{cases} |\boldsymbol{r}_i|/\xi_j & \text{if } 1 \leq |\boldsymbol{r}_i| < \xi_j, \\ 1 & \text{if } |\boldsymbol{r}_i| \geq \xi_j. \end{cases} \tag{9.3.2}$$

where ξ_j *is a fixed characteristic behaviour parameter of the host* $\boldsymbol{c}_j$*, called the* route length threshold.

Lemma 9.8. *Let* PCB_{ij} *denote the PCB for a malicious host* $\boldsymbol{c}_j$ *when processing an agent with route* $\boldsymbol{r}_i$*. Then:*

$$\mathrm{PCB}_{ij} = \pi_j \left(\frac{\min(|\boldsymbol{r}_i|, \xi_j)}{\xi_j} \right) . \tag{9.3.3}$$

Proof. From (A-6) we have $\mathrm{PCB}_{ij} = \mathrm{PIA}_{ij} \times \mathrm{PMM}_j$. The result then follows immediately from Definitions 9.6 and 9.7. □

The following remarks follow trivially from Lemma 9.8 and (A-6).

Remark 9.9. *Let* PCB_{ij} *denote the PCB for a malicious host* $\boldsymbol{c}_j$ *when processing an agent with route* $\boldsymbol{r}_i$*. The probability that* $\boldsymbol{c}_j$ *does not misbehave is* $1 - \mathrm{PCB}_{ij}$.

Remark 9.10. *Suppose that $\boldsymbol{c}_j$ and $\boldsymbol{c}_j{}'$ are malicious hosts that are visited by spy agents $\boldsymbol{r}_i$ and $\boldsymbol{r}_i{}'$, respectively, where $\boldsymbol{c}_j$ and $\boldsymbol{r}_i$ are not necessarily different from $\boldsymbol{c}_j{}'$ and $\boldsymbol{r}_i{}'$, respectively. Whether or not $\boldsymbol{c}_j$ processes $\boldsymbol{r}_i$ maliciously is independent of whether or not $\boldsymbol{c}_j{}'$ processes $\boldsymbol{r}_i{}'$ maliciously.*

In line with (A-8), we define PUR as follows.

Definition 9.11. *The PUR of a spy agent route $\boldsymbol{r}_i$ (written PUR_i) is the probability that $\boldsymbol{r}_i$ contains a non-empty set A of malicious hosts and each host in the set A does not misbehave (i.e. $\boldsymbol{r}_i$ gives a negative result).*

Lemma 9.12. *Let PUR_i denote the PUR for route $\boldsymbol{r}_i$, and let $\Pr(A, i)$ be the probability that the set of malicious hosts in $\boldsymbol{r}_i$ is A. Then:*

$$\mathrm{PUR}_i = \sum_{A \subseteq \boldsymbol{r}_i, A \neq \emptyset} \Pr(A, i) \prod_{\boldsymbol{c}_j \in A} (1 - \mathrm{PCB}_{ij}) \,. \tag{9.3.4}$$

Proof. From Definition 9.11 we have:

$$\mathrm{PUR}_i = \sum_{A \subseteq \boldsymbol{r}_i, A \neq \emptyset} \Pr(A, i) \Pr(\boldsymbol{c}_j \text{ does not misbehave for every } \boldsymbol{c}_j \in A) \,.$$

The result follows immediately from Remarks 9.9 and 9.10. □

Corollary 9.13. *If a route $\boldsymbol{r}_i$ contains no malicious hosts, then $\mathrm{PUR}_i = 0$.*

Proof. This follows immediately from the definition of PUR, since if $\boldsymbol{r}_i$ contains no malicious hosts, then $\Pr(A, i) = 0$. □

Corollary 9.14. *If a route $\boldsymbol{r}_i$ contains at least one malicious host that behaves maliciously then $\mathrm{PUR}_i = 0$.*

Proof. This follows immediately from the definition of PUR, since if $\boldsymbol{r}_i$ contains at least one malicious host, $\boldsymbol{c}_j$, that behaves maliciously then $\mathrm{PCB}_{ij} = 1$. Moreover, if A is a subset of $\boldsymbol{r}_i$ that does not contain $\boldsymbol{c}_j$ then $\Pr(A, i) = 0$. □

One problem with applying Lemma 9.12 is that the probability $\Pr(A, i)$ is generally not known, as the operator of a spy agent system will not a priori know the probability distribution of malicious hosts. We address this by estimating this probability, using the following result.

Lemma 9.15. *Suppose that a set of n hosts contains d malicious hosts, $0 \leq d \leq n$, where each host is equally likely to be malicious. Suppose that route $\boldsymbol{r}_i$ contains r hosts ($1 \leq r \leq n$). Then the probability $\Pr(r, \theta, n, d)$ that $\boldsymbol{r}_i$ contains exactly θ malicious hosts (where $0 \leq \theta \leq \min\{r, d\}$ and $r \leq n - d + \theta$) is given by:*

$$\Pr(r, \theta, n, d) = \binom{d}{\theta}\binom{n-d}{r-\theta} \Big/ \binom{n}{r}. \tag{9.3.5}$$

Proof. There are $\binom{d}{\theta}$ ways of choosing the θ malicious hosts in $\boldsymbol{r}_i$. Also, since there are $n - d$ non-malicious hosts, there are $\binom{n-d}{r-\theta}$ ways of choosing the $r - \theta$ non-malicious hosts in $\boldsymbol{r}_i$. There are $\binom{n}{r}$ possible routes containing r hosts. The result follows. □

By convention we set $\Pr(r, \theta, n, d) = 0$ if $\theta > d$, $\theta > r$, or $n - d < r - \theta$.

The following remark follows immediately from the definition of $\Pr(r, \theta, n, d)$.

Remark 9.16. *If $\theta \leq r$, $d \leq n$, and $r \leq n$, then*

$$\sum_{\theta=0}^{r} \Pr(r, \theta, n, d) = 1. \tag{9.3.6}$$

The following two corollaries follow trivially.

Corollary 9.17. *Suppose that a set S of n hosts contains d malicious hosts. The probability that an r-subset of S contains no malicious hosts is given by:*

$$\Pr(r, 0, n, d) = \begin{cases} \binom{n-d}{r} \Big/ \binom{n}{r} = \prod_{l=0}^{r-1} \frac{n-d-l}{n-l} & \text{if } r \leq n - d, \\ 0 & \text{otherwise.} \end{cases} \tag{9.3.7}$$

Corollary 9.18. *Suppose that a set S of n hosts contains d malicious hosts. The probability that an r-subset of S contains r malicious hosts is given by:*

$$\Pr(r, r, n, d) = \begin{cases} \binom{d}{r}/\binom{n}{r} = \prod_{k=0}^{k=r-1} \frac{d-k}{n-k} & \text{if } r \leq d, \\ 0 & \text{otherwise.} \end{cases} \tag{9.3.8}$$

Using Lemma 9.15, we can obtain the following estimate for PUR.

Lemma 9.19. *Suppose that a set of n hosts contains d malicious hosts, $0 \leq d \leq n$, where each host is equally likely to be malicious. Suppose that route $\boldsymbol{r}_i$ contains r hosts ($1 \leq r \leq n$). Then, using the notation of Lemma 9.12:*

$$\mathrm{PUR}_i = \sum_{\theta=1}^{r} \Pr(r, \theta, n, d) \binom{r}{\theta}^{-1} \sum_{A \subseteq \boldsymbol{r}_i, |A|=\theta} \left(\prod_{\boldsymbol{c}_j \in A} (1 - \mathrm{PCB}_{ij}) \right) . \tag{9.3.9}$$

Proof. Let $\Pr(A, i)$ be as defined in Lemma 9.12, i.e. the probability that A is the set of malicious hosts in $\boldsymbol{r}_i$. Then, if $|A| = \theta$, it follows by definition that

$$\Pr(A, i) = \Pr(r, \theta, n, d) / \binom{r}{\theta} .$$

Now, from Lemma 9.12:

$$\mathrm{PUR}_i = \sum_{\theta=1}^{r} \left(\sum_{A \subseteq \boldsymbol{r}_i, |A|=\theta} \Pr(A, i) \prod_{\boldsymbol{c}_j \in A} (1 - \mathrm{PCB}_{ij}) \right)$$

and hence

$$\mathrm{PUR}_i = \sum_{\theta=1}^{r} \Pr(r, \theta, n, d) \binom{r}{\theta}^{-1} \sum_{A \subseteq \boldsymbol{r}_i, |A|=\theta} \left(\prod_{\boldsymbol{c}_j \in A} (1 - \mathrm{PCB}_{ij}) \right) ,$$

as desired. □

Lemma 9.19, when combined with Lemma 9.15, enables us to estimate the PUR of a route when the parameters n, d, r, π_j, ξ_j are known.

Remark 9.20. *By definition, the PUR of one route is independent of the PUR of any other route.*

We next use the notion of PUR to evaluate the CC of a spy agent classification outcome, in line with (A-9).

Lemma 9.21. *Let $\mathcal{R}$ be a set of spy agent routes that is used to test a set of hosts, and let $\mathcal{R}_N \subseteq \mathcal{R}$ be the set of routes that yield a negative result. Then, using the notation of Lemma 9.19, the* CC *of this spy agent scheme is:*

$$\text{CC} = \prod_{\boldsymbol{r}_i \in \mathcal{R}_N} (1 - \text{PUR}_i)\,. \tag{9.3.10}$$

Proof. The outcome set of this scheme is credible if and only if the outcome of every route $\boldsymbol{r}_i \in \mathcal{R}$ is credible. From Definition 9.11 and Remark 9.20 it follows that:

$$\begin{aligned} \text{CC} &= \prod_{\boldsymbol{r}_i \in \mathcal{R}} (1 - \text{PUR}_i) \\ &= \prod_{\boldsymbol{r}_i \in \mathcal{R}_N} (1 - \text{PUR}_i) \prod_{\boldsymbol{r}_j \in (\mathcal{R} - \mathcal{R}_N)} (1 - \text{PUR}_j)\,. \end{aligned}$$

The result follows from Corollary 9.13. □

In general, the notion of CC is useful for benchmarking the effectiveness of spy agent route designs. We note, however, that the above result does not capture how the an uncharacteristic outcome affects host identification results. That is, the CC does not take account of the ability of some spy agent schemes to successfully classify a set of hosts even in the presence of uncharacteristic results. One example of such a spy agent scheme was proposed in Chapter 7, where in some cases malicious hosts can be correctly identified even if they do not always misbehave. Further, the CC does not take account of the fact that certain types of host behaviour can give inconsistent or misleading spy agent results, as discussed in §9.2.2 (see Examples 9.1–9.5). The study of more complex evaluation models remains a topic for future work.

We next demonstrate the use of the above results by providing a detailed analysis of four case studies.

9.3.3 Case study 1: Single spy agent route

In our first case study we consider a simple scenario in which a single spy agent migrates through a set of r hosts, i.e. the route design is $\mathcal{R} = \{\boldsymbol{r}_1 = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_r\}\}$. We further assume that all malicious hosts contained in the route have the same PMM and PIA, i.e. $\pi_j = \pi$ and $\xi_j = \xi$ for $1 \leq j \leq r$. If $\boldsymbol{r}_1$ contains θ malicious hosts, then these hosts can be distributed in $\binom{r}{\theta}$ ways. Combining Lemmas 9.8, 9.19 and 9.21 we obtain the following estimate for the CC of the system (written $\mathrm{CC_S}$).

$$\begin{aligned}
\mathrm{CC_S} &= \prod_{\boldsymbol{r}_i \in \mathcal{R}} (1 - \mathrm{PUR}_i) \\
&= 1 - \mathrm{PUR}_1 \\
&= 1 - \sum_{\theta=1}^{r} \Pr(r, \theta, n, d) \binom{r}{\theta}^{-1} \sum_{k=1}^{\binom{r}{\theta}} \prod_{j=1}^{\theta} (1 - \pi(\min(r, \xi))/\xi) \\
&= 1 - \sum_{\theta=1}^{r} \Pr(r, \theta, n, d) \left(1 - \pi(\min(r, \xi))/\xi\right)^{\theta} . \qquad (9.3.11)
\end{aligned}$$

Remark 9.22. *In the above equation* $\mathrm{CC_S} = 1$ *if and only if* $r \geq \xi$ *and* $\pi = 1$*, i.e. there is at least one malicious host and it always behaves maliciously.*

In Figures 9.2–9.9 we illustrate the effect on $\mathrm{CC_S}$ of a range of values for the system parameters. We make the following observations.

(GEN) In the figures we use a colour ramp palette that interpolates between 'red' and 'green', and we associate the colours of this palette with the values of $\mathrm{CC_S}$, which range from 0 (red) to 1 (green).

1. $\mathrm{CC_S}$ is a function of five parameters: the route length r, the number

of malicious hosts, d, the number of target hosts, n, the route length threshold of malicious hosts, ξ, and the malicious host PMM, π.

2. We consider all possible cases where three of the four parameters d, n, ξ, π are fixed (we do not consider the case where r is fixed). This gives four groups of evaluation results, **(A)**–**(D)**, which are discussed below.

(A) Figures 9.2 and 9.3 plot CC_S against $r \times \pi$, $1 \leq r \leq n$, $0 \leq \pi \leq 1$, for fixed d, n, ξ. We first observe that, in all these figures, CC_S increases monotonically with π. In all the contour maps, greener colours are obtained (meaning $CC_S \to 1$) as the PMM π increases. We also observe that the effect of r is non-convex.

A visual comparison suggests that CC_S tends to increase as r increases, with the following three exceptions: 1) if $\pi \to 0$, then $CC_S \to 0$; 2) if $\pi \to 1$, then CC_S has two local maxima and one minimum for a value of r which depends on the remaining three parameters; and 3) for smaller values of d, the minimum value of CC_S in exception 2) is obtained for a larger r (the red zone shifts to the right). We note that, while r is determined by the route design, π depends on inherent behavioural characteristics of the malicious hosts. This emphasises the importance of making accurate assumptions regarding malicious host behaviour.

We make the following additional observations regarding changes to the other parameters.

1. Figures 9.2(c) and 9.2(d) plot CC_S for smaller and larger values of d, respectively, than the value used in Figure 9.2(b). We observe that, as d increases, so does CC_S. One way of interpreting this

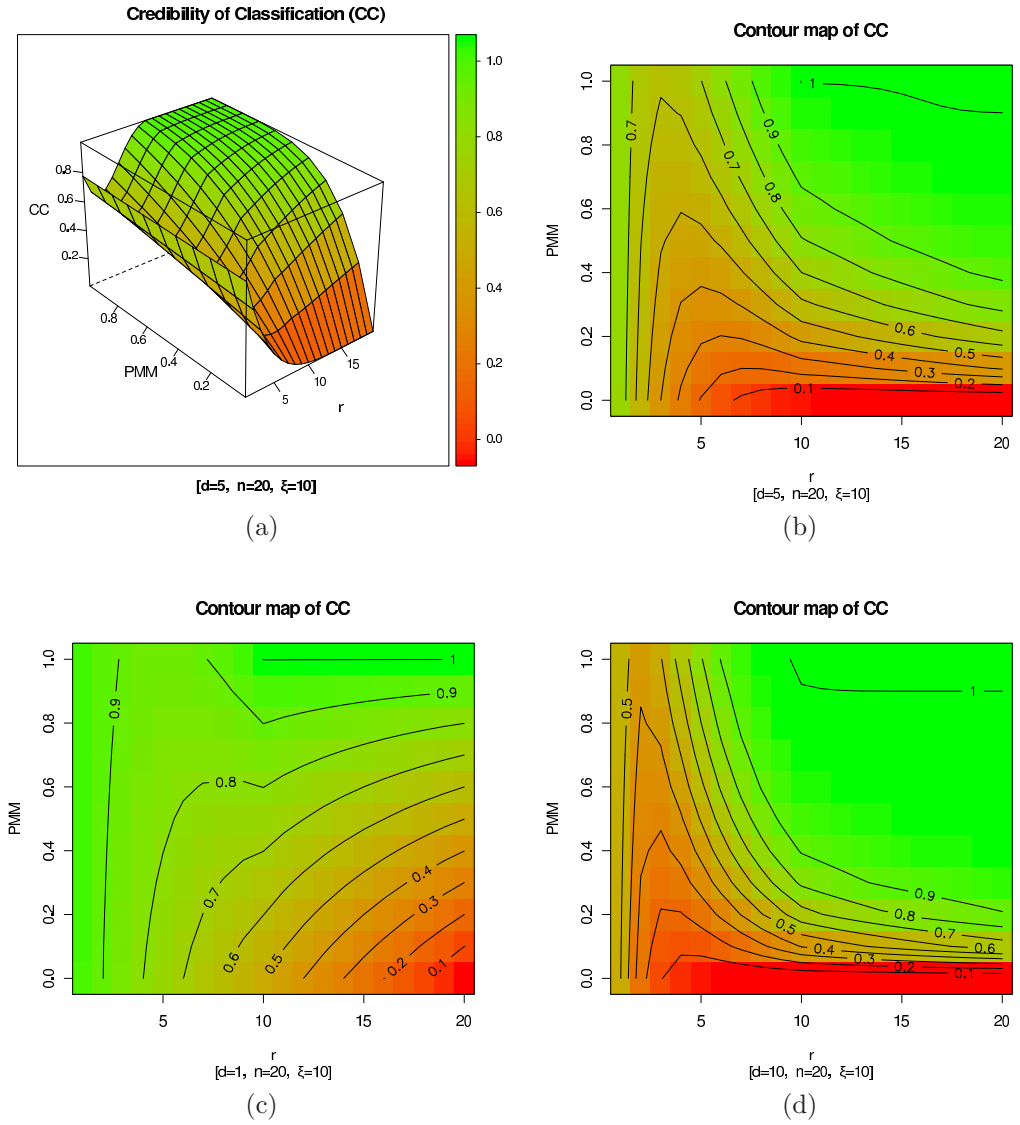
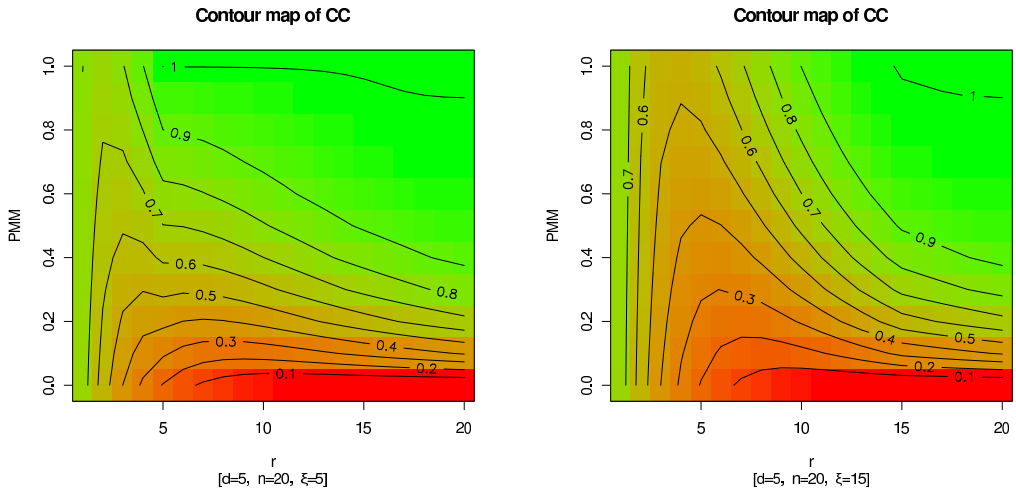
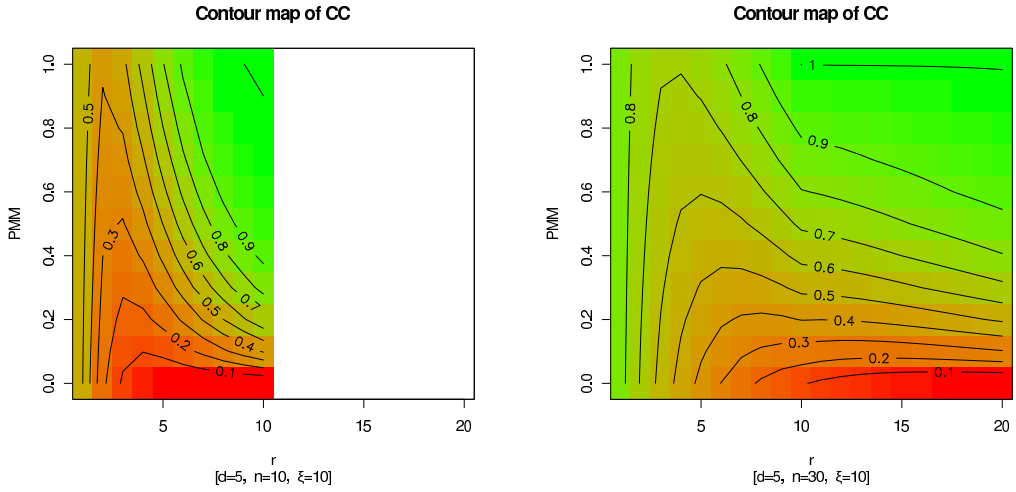


Figure 9.2: CC_S vs. $r \times \pi$, for $r \in [1, 20]$ and $\pi \in [0, 1]$, for the cases where: (a),(b) $d = 5$, $n = 20$, and $\xi = 10$; (c) $d = 1$, $n = 20$, and $\xi = 10$; and (d) $d = 10$, $n = 20$, and $\xi = 10$.



(a) (b)



(c) (d)

Figure 9.3: CC_S vs. $r \times \pi$, for $r \in [1, 20]$ and $\pi \in [0, 1]$, for the cases where: (a) $d = 5$, $n = 20$, and $\xi = 5$; (b) $d = 5$, $n = 20$, and $\xi = 15$; (c) $d = 5$, $n = 10$, and $\xi = 10$; and (d) $d = 5$, $n = 30$, and $\xi = 10$.

result involves considering (9.3.5): a larger value of d increases the probability that a route contains a larger set of malicious hosts, which in turn reduces the probability that a route giving a negative result contains a set of malicious hosts, none of which misbehaves. This suggests that a spy agent outcome is more credible when more hosts are found (or known) to be malicious.

2. Figures 9.3(a) and 9.3(b) plot $\mathrm{CC_S}$ for smaller and larger values of ξ, respectively, than the value used in Figure 9.2(b). We observe that $\mathrm{CC_S}$ increases monotonically as ξ decreases. This effect arises from our assumption that a malicious host is more likely to behave maliciously if it has a smaller route length threshold ξ, as follows from (9.3.3).

3. Figures 9.3(c) and 9.3(d) plot $\mathrm{CC_S}$ for smaller and larger values of n, respectively, than the value used in Figure 9.2(b). A visual comparison suggests that by, as n increases, $\mathrm{CC_S}$ decreases. This effect is further analysed under **(D)** below.

(B) Figures 9.4 and 9.5 plot $\mathrm{CC_S}$ against $r \times \xi$, $1 \leq r \leq n$, $0 \leq \xi \leq n$, for fixed d, n, π. We first observe that, in all these figures, $\mathrm{CC_S}$ increases monotonically as ξ decreases. This observation is consistent with **(A-2)** above. We further observe that, if $r \geq \xi$, then changes in ξ have no effect on $\mathrm{CC_S}$. This is because, in this case, the value of PCB, as given in (9.3.3), is independent of ξ. As in **(A)** above, the effect of r is non-convex.

We make the following additional observations regarding changes to the other parameters.

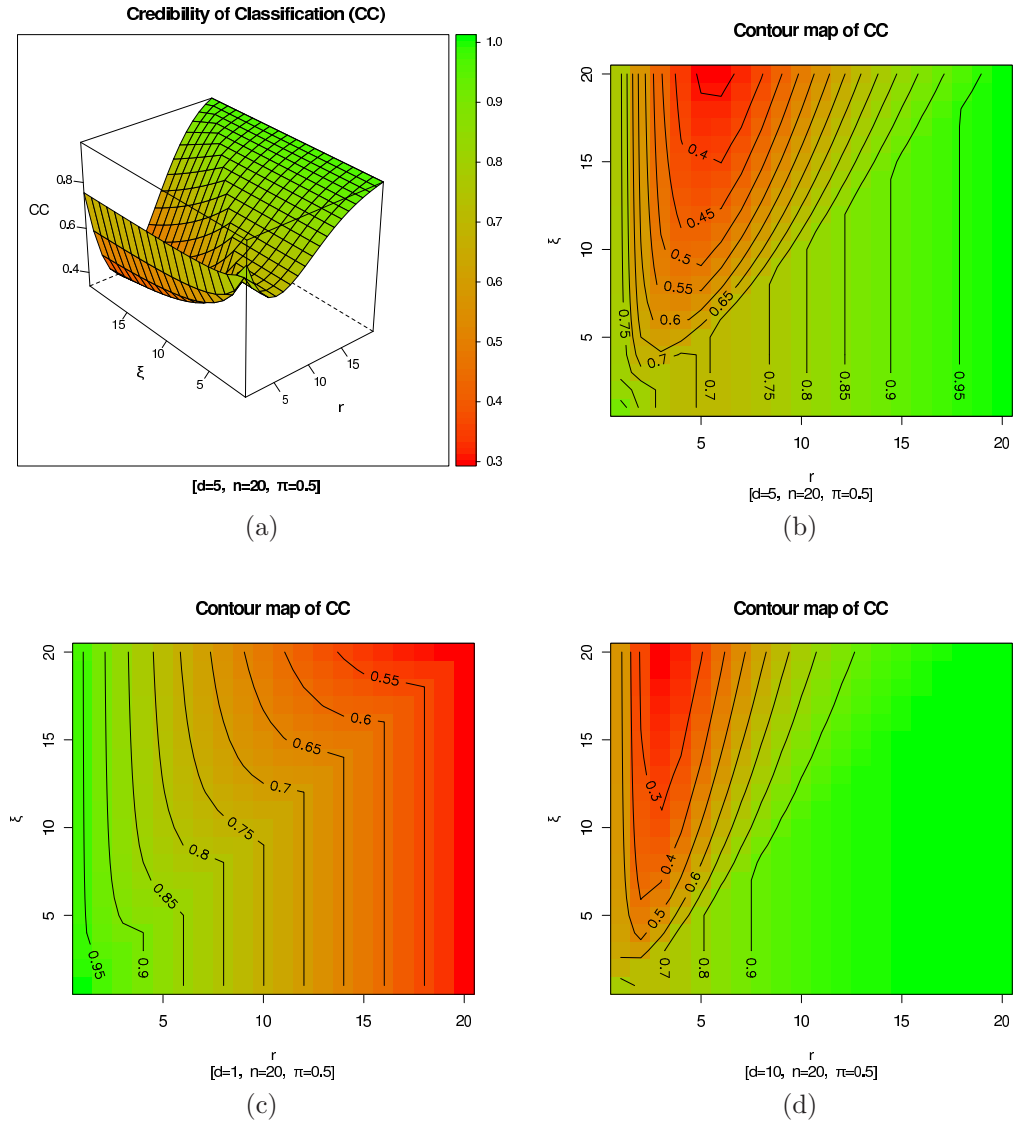


Figure 9.4: CC_S vs. $r \times \xi$, for $r \in [1, 20]$ and $\xi \in [0, 1]$, for the cases where: (a),(b) $d = 5$, $n = 20$, and $\pi = 0.5$; (c) $d = 1$, $n = 20$, and $\pi = 0.5$; and (d) $d = 10$, $n = 20$, and $\pi = 0.5$.

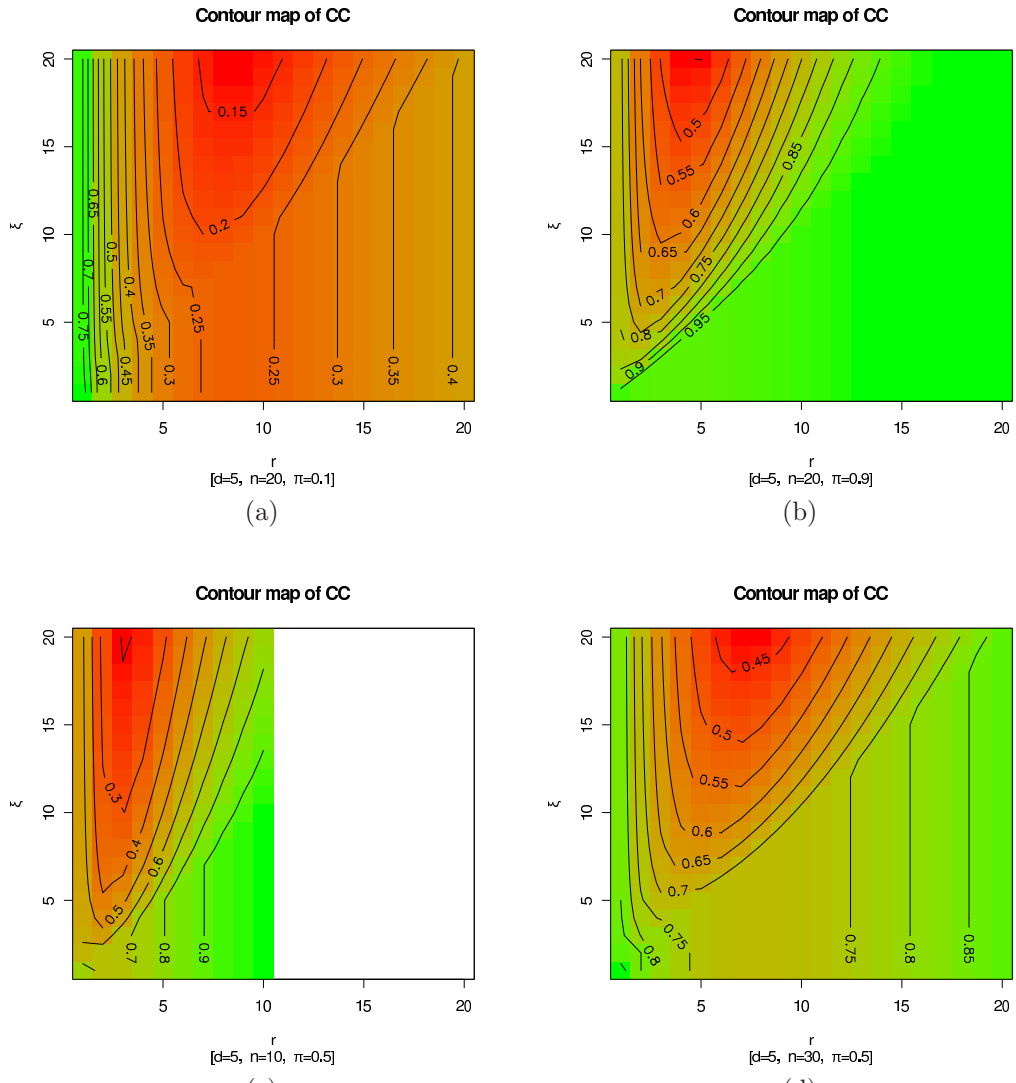


Figure 9.5: CC_S vs. $r \times \xi$, for $r \in [1, 20]$ and $\xi \in [0, 1]$, for the cases where: (a) $d = 5$, $n = 20$, and $\pi = 0.1$; (b) $d = 5$, $n = 20$, and $\pi = 0.9$; (c) $d = 5$, $n = 10$, and $\pi = 0.5$; and (d) $d = 5$, $n = 30$, and $\pi = 0.5$.

1. Figures 9.4(c) and 9.4(d) plot CC_S for smaller and larger values of d, respectively, than the value used in Figure 9.4(b). We observe that, as d increases, so does CC_S. This result is consistent with the analysis in **(A-1)** above.

2. Figures 9.5(a) and 9.5(b) plot CC_S for smaller and larger values of π, respectively, than the value used in Figure 9.4(b). We observe that relatively large values of CC_S are obtained with large π (that is, for the case where malicious hosts are more likely to misbehave). In this case, relatively large values of CC_S are obtained for larger values of r, regardless of the other parameters. This behaviour arises because a malicious host is very likely to behave maliciously when $\pi \to 1$ and when PIA $\to 1$, as follows from (9.3.2). This conclusion reinforces the conclusions reached under **(A)** above.

3. Figures 9.5(c) and 9.5(d) plot CC_S for smaller and larger values of n, respectively, than the value used in Figure 9.4(b). A visual comparison suggests that, as n increases, CC_S decreases. This effect is further analysed under **(D)** below.

(C) Figures 9.6 and 9.7 plot CC_S against $r \times d$, $1 \leq r \leq n$, $1 \leq d \leq n$, for fixed n, ξ, π. In this case CC_S exhibits two local minima. One arises for a large value of d and a small value of r, and the other for a small value of d and a large value of r. This extends the observations made under **(A)** above where, for a fixed d, CC_S exhibits one local minimum. These observations reinforce the conclusions given in **(A-1)**. For example, the local minimum (red area) in Figure 9.2(c) appears on the right side of the map (large r, small d), and the local minimum in Figure 9.2(d) appears

on the left side (small r, large d).

This behaviour emphasises the importance of choosing an appropriate value of r for a spy agent routing scheme. This choice becomes more difficult when there is no a priori knowledge of d. In such cases, it would appear prudent to choose a relatively moderate value for r, i.e. neither too high nor too low. The initial choice can be further refined as knowledge builds up (e.g. with the help of previous spy agent evaluations). For example, previously obtained results might suggest that a larger value of r should be used in order to improve CC_S if there are a large number of malicious hosts, and/or if the malicious hosts are more likely to exhibit malicious behaviour. If, however, there are only a very small number of malicious hosts, and if these hosts have a very small PMM, then the probability that a route contains a malicious host that does not misbehave is likely to be higher for a larger route length.

A visual comparison of Figures 9.6 and 9.7 further suggests that CC_S is more 'unstable' for small d, i.e. its gradient is larger; that is, it changes faster as other parameters change. We make the following additional observations regarding changes to the other parameters.

1. Figures 9.6(c) and 9.6(d) plot CC_S for larger and smaller values of n, respectively, than the value used in Figure 9.6(b). A visual comparison suggests that, as n increases, CC_S decreases. This effect is consistent with **(A-3)** and **(B-3)** above, and this issue is further analysed under **(D)** below.

2. Figures 9.7(a) and 9.7(b) plot CC_S for smaller and larger values of π, respectively, than the value used in Figure 9.6(b). These plots

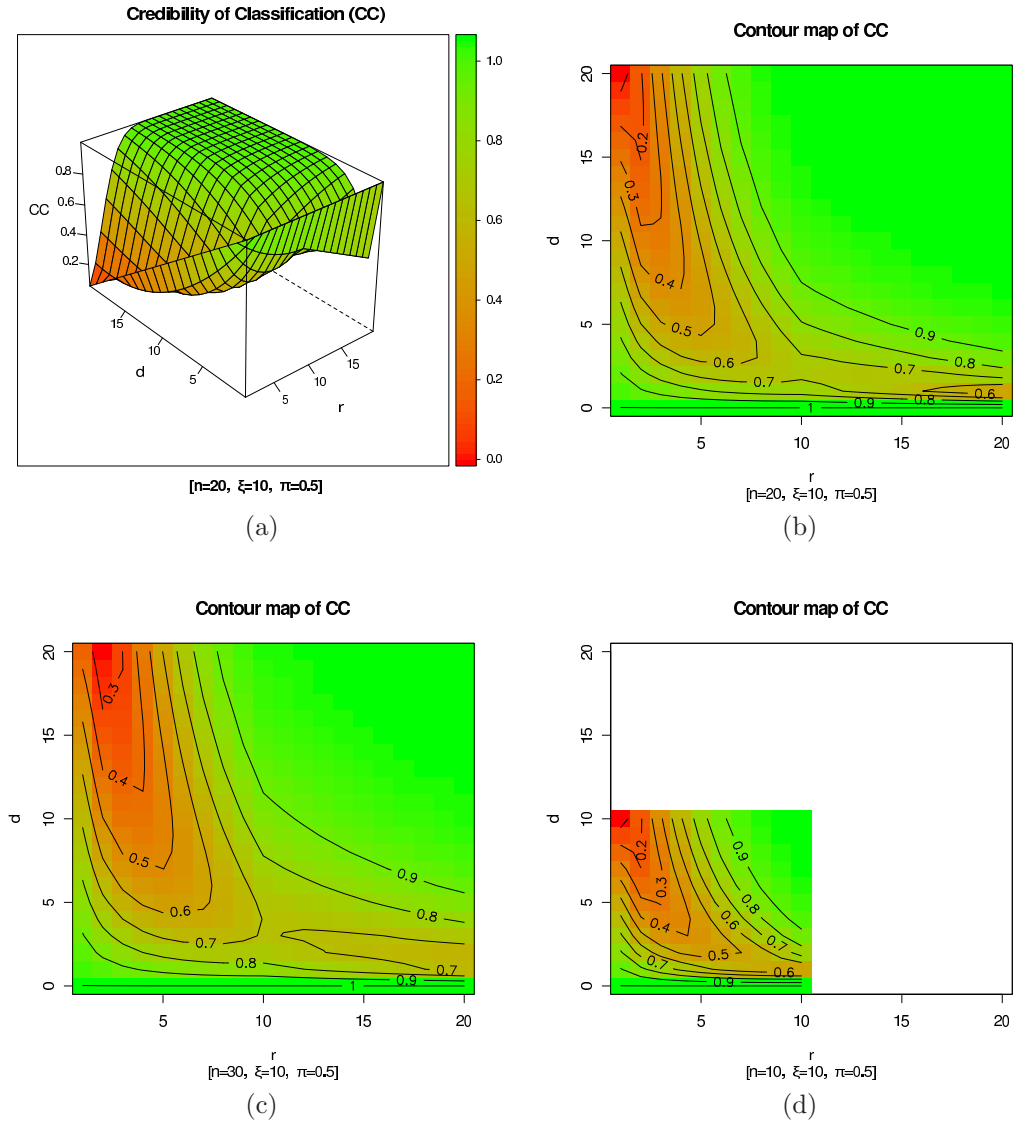


Figure 9.6: CC_S vs. $r \times d$, for $r \in [1, 20]$ and $d \in [0, 20]$, for the cases where: (a),(b) $n = 20$, $\xi = 10$, and $\pi = 0.5$; (c) $n = 30$, $\xi = 10$, and $\pi = 0.5$; and (d) $n = 10$, $\xi = 10$, and $\pi = 0.5$.

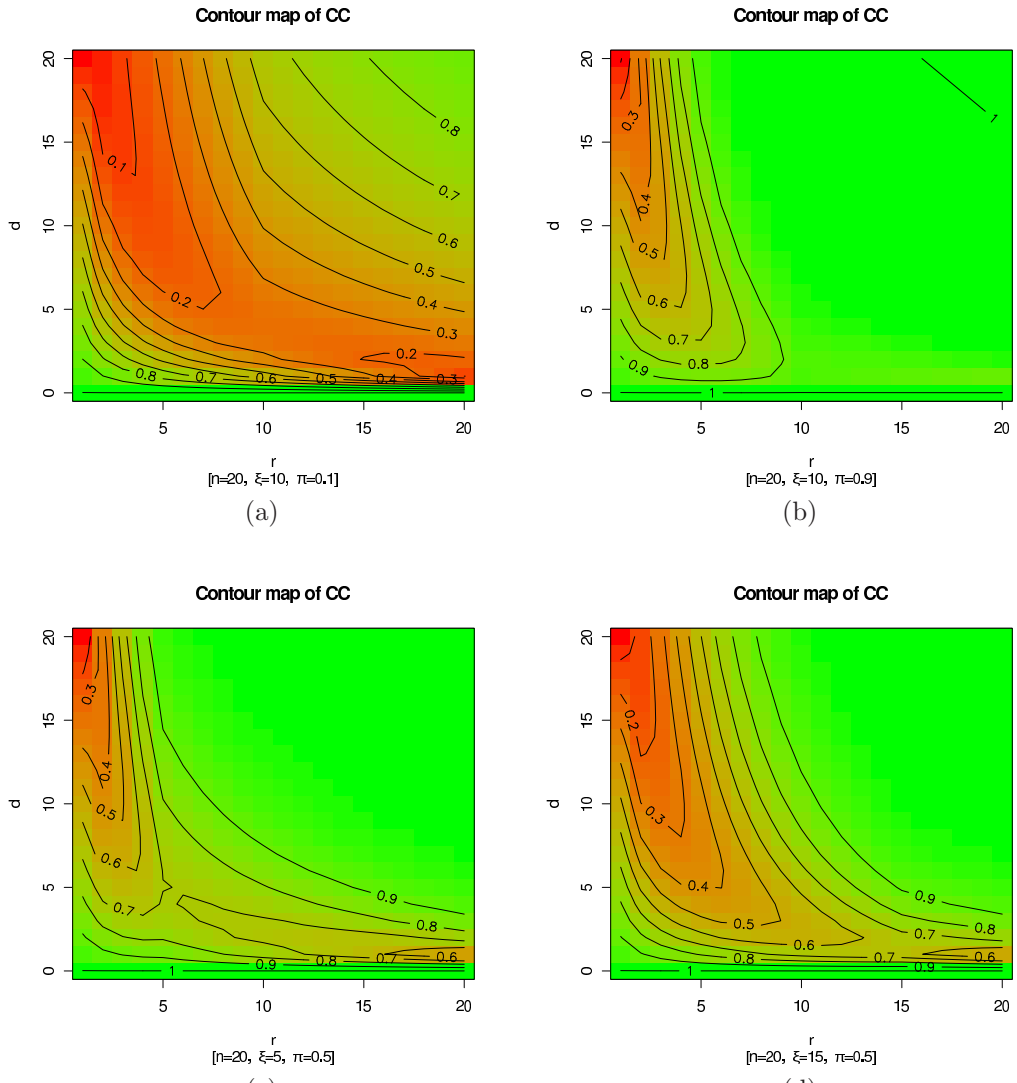


Figure 9.7: CC_S vs. $r \times d$, for $r \in [1, 20]$ and $d \in [0, 20]$, for the cases where: (a) $n = 20$, $\xi = 10$, and $\pi = 0.1$; (b) $n = 20$, $\xi = 10$, and $\pi = 0.9$; (c) $n = 20$, $\xi = 5$, and $\pi = 0.5$; and (d) $n = 20$, $\xi = 15$, and $\pi = 0.5$.

are consistent with the discussion under **(A)** above.

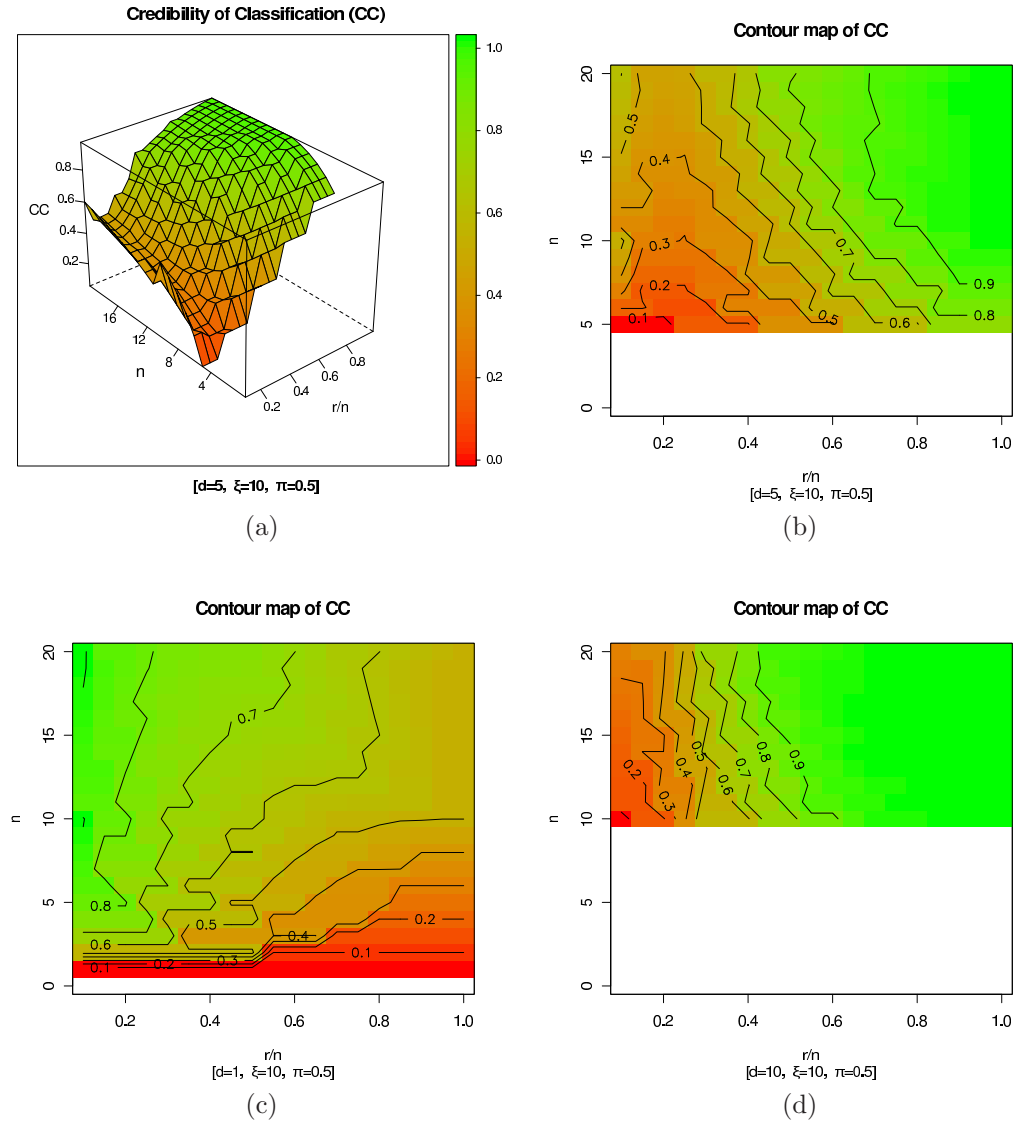
3. Figures 9.7(c) and 9.7(d) plot CC_S for smaller and larger values of ξ, respectively, than the value used in Figure 9.6(b). These plots are consistent with the discussion in **(A-2)** above.

(D) Figures 9.8 and 9.9 plot CC_S against $\frac{r}{n} \times n$, $0 \leq \frac{r}{n} \leq 1$, $1 \leq n \leq 20$, for fixed d, ξ, π. We first make two general observations. 1) If the ratio r/n is kept fixed, increases in n seem to have a relatively small effect on CC_S, although there are some exceptions. That is, the contour lines are predominantly vertical in the corresponding plots. 2) If π and d are both relatively large, high values of CC_S are obtained when $r/n \to 1$ (i.e. when r is maximised).

Combining these observations with the discussions in **(A-3)**, **(B-3)**, and **(C-1)** above, we suggest that, if n is increased while keeping all other parameters fixed, the exhibited decrease in CC_S can mainly be attributed to the decrease in the ratio r/n (rather than the increase in n). This underlines the importance of selecting spy agent routes with long migration paths.

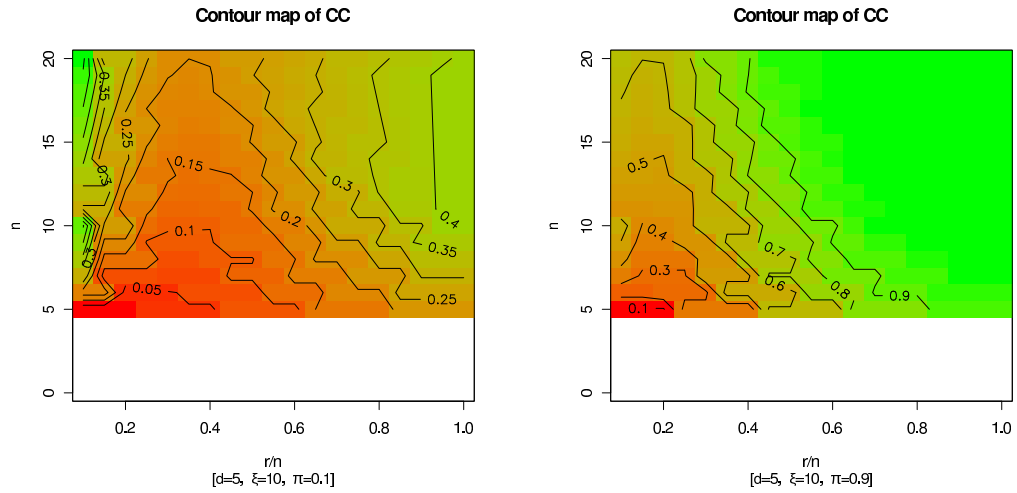
However, as indicated above, there are a few exceptions to this picture. That is, CC_S takes a relatively low value (as marked in red) when π or d are small. This behaviour is consistent with previous observations. By only considering the case where π is large, the findings suggest that the model presented in this chapter is more appropriate if malicious hosts have a high likelihood of behaving as expected (i.e. maliciously).

We make the following additional observations regarding changes to the other parameters.



(a) (b) (c) (d)

Figure 9.8: CC_S vs. $\frac{r}{n} \times n$, for $\frac{r}{n} \in [0, 1]$ and $n \in [0, 20]$, for the cases where: (a),(b) $d = 5$, $\xi = 10$, and $\pi = 0.5$; (c) $d = 1$, $\xi = 10$, and $\pi = 0.5$; and (d) $d = 10$, $\xi = 10$, and $\pi = 0.5$.



(a) (b) (c) (d)

Figure 9.9: CC_S vs. $\frac{r}{n} \times n$, for $\frac{r}{n} \in [0, 1]$ and $n \in [0, 20]$, for the cases where: (a) $d = 5$, $\xi = 10$, and $\pi = 0.1$; (b) $d = 5$, $\xi = 10$, and $\pi = 0.9$; (c) $d = 5$, $\xi = 5$, and $\pi = 0.5$; and (d) $d = 5$, $\xi = 15$, and $\pi = 0.5$.

1. Figures 9.8(c) and 9.8(d) plot CC_S for smaller and larger values of d, respectively, than the value used in Figure 9.8(b). We observe that, as d increases, CC_S reaches a maximum for a relatively large value of r/n. This observation supports the conclusions given under **(D)** above.

2. Figures 9.9(a) and 9.9(b) plot CC_S for smaller and larger values of π, respectively, than the value used in Figure 9.8(b). As for d, as π increases, CC_S reaches a maximum for a relatively large value of r/n. This observation also supports the conclusions given under **(D)** above.

3. Figures 9.9(c) and 9.9(d) plot CC_S for smaller and larger values of ξ, respectively, than the value used in Figure 9.8(b). We observe that ξ significantly influences the results only when $r < \xi$. For larger values of r, the effect of ξ is diminished, yielding more credible results.

We note that the above observations will also apply for more complex route designs. This is because from Remark 9.20 we observe that the CC_S of individual routes are independent of one another. This is further discussed in the next case study.

9.3.4 Case study 2: A homogeneous system

In our second case study we consider the special case where all routes have the same length, r, and the PMM and PIA of every malicious host $\boldsymbol{c}_j$ is the same, i.e. $\pi_j = \pi$ and $\xi_j = \xi$. The analysis of this case builds upon the first case study, and the CC of this system (written CC_H) can be trivially derived

from Lemma 9.21 and (9.3.11) as:

$$\mathrm{CC_H} = (\mathrm{CC_S})^{\beta}\,, \tag{9.3.12}$$

where $\beta = |\mathcal{R}_N|$ is the number of routes that yield a negative result.

This case study applies to both NGT and SGT spy agent route designs, as described in Chapters 6 and 8, respectively. Specific studies of these two types of route design are given below.

9.3.5 Case study 3: NGT route designs

The CC of a NGT spy agent routing design can be estimated from (9.3.12) in the case where a design with uniform route length is used. For example, this is the case if the spy agent routes are constructed using a t-$(v, b, r, k, 1)$ design, as discussed in §6.4.2. This is achieved by associating the blocks and points of the design with hosts and routes, respectively. In this case, there are v spy agents, b hosts, k agents visiting each host and r hosts per agent.

In Figures 9.10 and 9.11 we plot estimates for the CC for certain well-known t-$(v, b, r, k, 1)$ designs. In these examples, the parameters r and n are determined by the design. From Theorem 6.19 and Corollary 6.18 we know that 2-designs are $k-1$-disjunct, and $\overline{k-1}$-classifiers. This means that the corresponding routing designs can only identify a limited number of malicious hosts. For this reason, we only consider small values of d, i.e. $1 \leq d \leq 4$. For convenience, we set $\pi = 0.7$ (giving rise to more credible results, as discussed in §9.3.3) and $\beta = 1$ (i.e. there is only spy agent that yields a negative result).

We make the following observations regarding these figures.

1. The value of CC is smaller for larger d and larger ξ.

2. The value of CC is larger for designs with larger r and smaller ξ. The value of CC is maximised when $r \geq \xi$. This means that 'larger' designs (such as the 2-$(27, 117, 13, 3, 1)$) will give more credible results.

These findings are consistent with the conclusions given in §9.3.3.

We note here that our choices for β and π may not be consistent with the choice of d. This is highlighted by the example that follows.

Example 9.23. Consider the design 2-$(7, 7, 3, 3, 1)$ in which a single malicious host is visited by $k = 3$ spy agents. If there is only one malicious host and this host abuses all visited agents, then three routes will yield a positive result, and $7 - 3 = 4$ routes will yield a negative result. This result is not consistent with the assumption $\beta = 1$. If, on the other hand, less than three routes give a positive result, then the outcome cannot be used to deterministically classify the hosts, and it is 100% certain that the outcome is not credible. This result is not consistent with our estimation of $\mathrm{CC_H}$.

More generally, as discussed in §9.2.2, the credibility model does not take into account whether or not a certain combination of spy agent results yields consistent host classification results. Instead, the simplifying assumption is that the PUR of a route is independent of the PUR of all other routes. For this reason, we claim that the credibility model may not be suitable for posterior evaluations of obtained results. Instead, it provides a technique that can be used to help choose route designs that are likely to give credible results.

9.3.6 Case study 4: SGT route designs

In this case study we wish to estimate the CC of the multi-stage sub-group routing algorithm described in §8.4. For convenience we only consider the

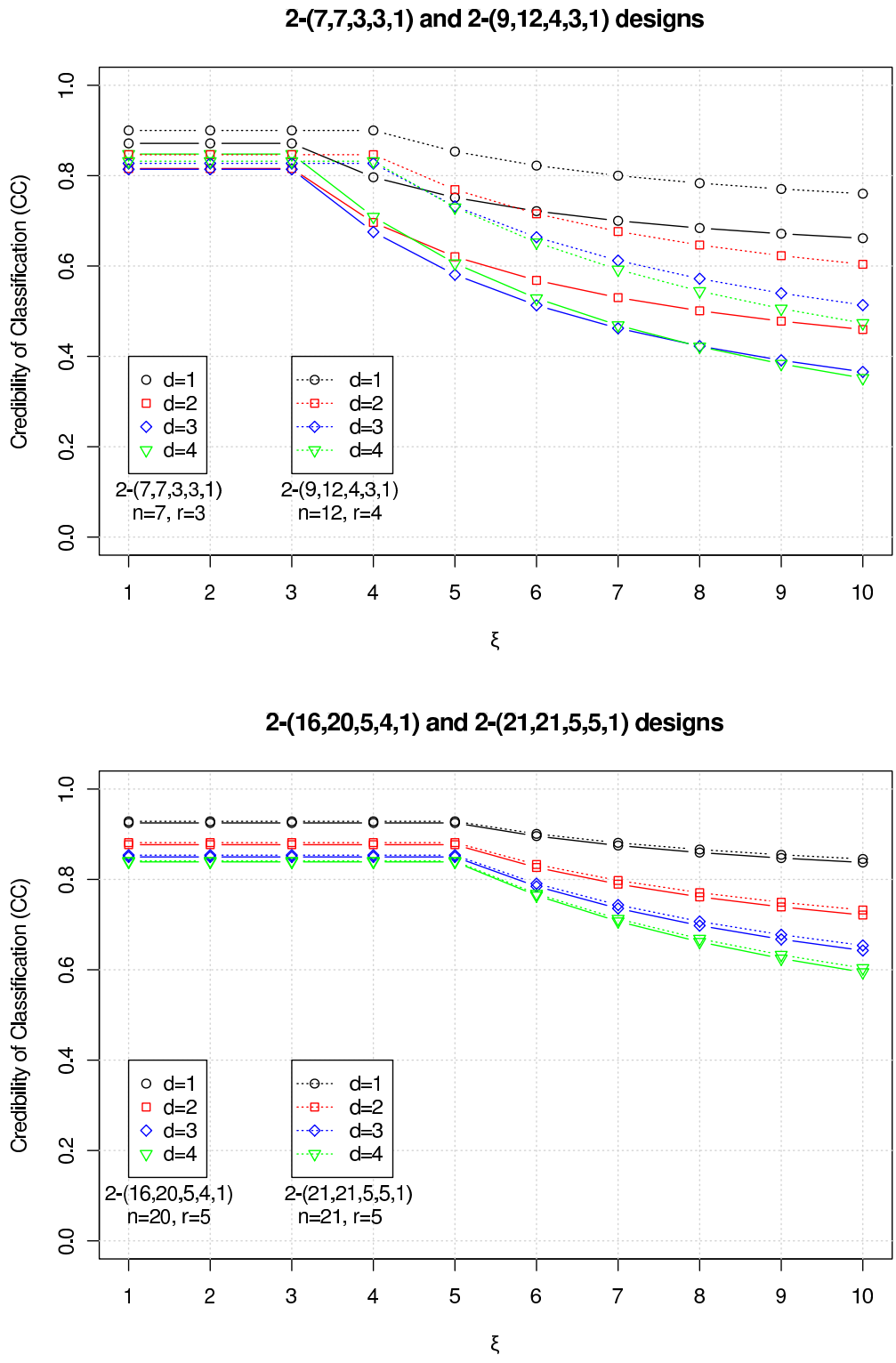


Figure 9.10: CC_S for 2-$(7, 7, 3, 3, 1)$, 2-$(9, 12, 4, 3, 1)$, 2-$(16, 20, 5, 4, 1)$ and 2-$(21, 21, 5, 5, 1)$ designs. We set $\pi = 0.9$.

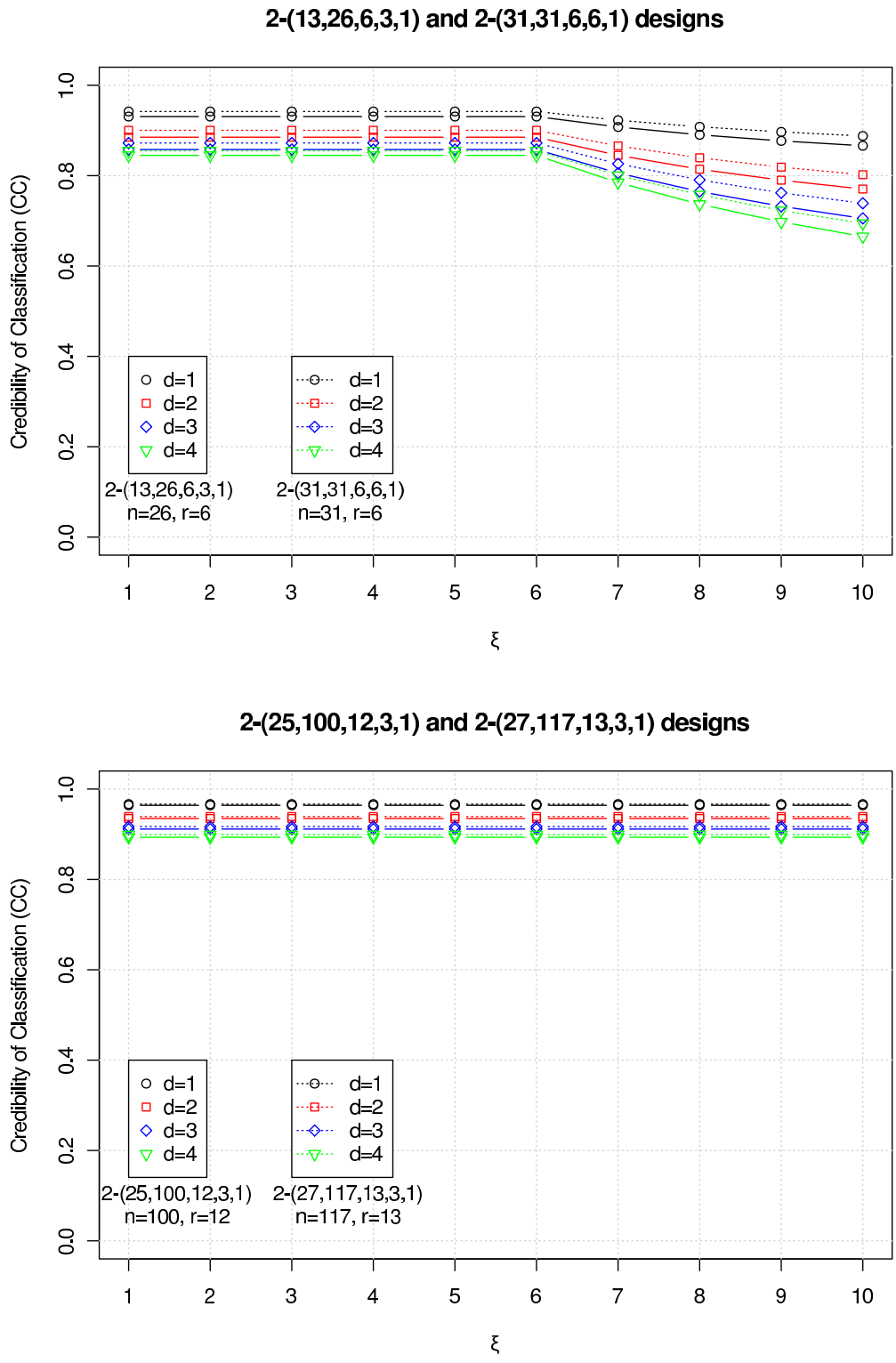


Figure 9.11: CC_S for 2-$(13, 26, 6, 3, 1)$, 2-$(31, 31, 6, 6, 1)$, 2-$(25, 100, 12, 3, 1)$ and 2-$(27, 117, 13, 3, 1)$ designs. We set $\pi = 0.9$.

case where the algorithm reaches a final stage in which malicious hosts are identified. In this case, if the algorithm tests n hosts and ends at stage i, then there is only one route that yields a negative result (i.e. $\beta = 1$), the single negative route has length $r = n - i$, and $d = i$ malicious hosts are identified. The CC of this scheme, written $\mathrm{CC_M}$, can then be trivially derived from (9.3.11) and (9.15) as follows.

$$\begin{aligned} \mathrm{CC_M} &= 1 - \sum_{\theta=1}^{n-d} \Pr(n-d, \theta, n, d) \left(1 - \pi(\min(n-d, \xi))/\xi\right)^{\theta} \\ &= 1 - \sum_{\theta=1}^{n-d} \frac{\binom{d}{\theta}\binom{n-d}{n-d-\theta}}{\binom{n}{n-d}} \left(1 - \frac{\pi(\min(n-d, \xi))}{\xi}\right)^{\theta}. \end{aligned} \tag{9.3.13}$$

Remark 9.24. *In the above equation* $\mathrm{CC_M} = 1$ *if and only if* $n \geq d + \xi$ *and* $\pi = 1$.

Remark 9.24 suggests that n should be chosen to be sufficiently large to allow higher values of $\mathrm{CC_M}$ to be obtained. We note that this result gives additional weight to the third part of Theorem 8.10, which states that Algorithm 8.9 is 'optimal' in the sense that it identifies malicious hosts using routes with the longest possible length.

$\mathrm{CC_M}$ is a function of four parameters: the number of target hosts, n, the number of malicious hosts, d, the route length threshold of malicious hosts, ξ, and the malicious host PMM, π. In Figure 9.12 we plot $\mathrm{CC_M}$ against $d \times \pi$, $1 \leq d \leq n$, $0 \leq \pi \leq 1$ for a case in which n and ξ are fixed. We make the following observations.

- For large values of d (large i, small r) the value of $\mathrm{CC_M}$ is small.
- For small values of d (large r) the value of $\mathrm{CC_M}$ depends on π. If, in this case, π is small (i.e. the malicious hosts behave less consistently) then the value of $\mathrm{CC_M}$ is small.

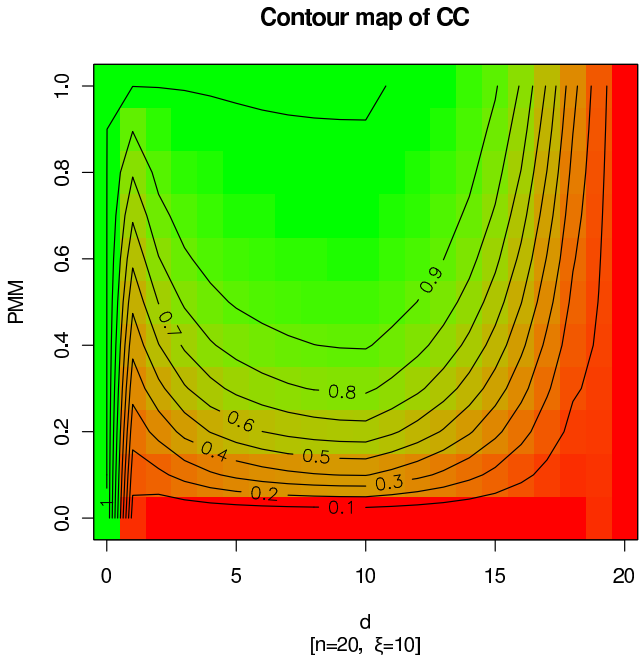


Figure 9.12: $\mathrm{CC_M}$ vs. $d \times \pi$, for $n = 20$, $\xi = 10$.

- The value of $\mathrm{CC_M}$ is more likely to be relatively large in the presence of neither too many nor too few malicious hosts.

The above results are consistent with the discussions in §9.3.3.

9.3.7 Discussion

The validity of the credibility model presented in this chapter depends on: a) the validity of the assumptions given in §9.3.1, b) the correctness of the estimates for the parameters d, ξ, and π, and c) the choice of the parameters n and r that characterise the route design, as discussed in §9.3.3.

In general, the credibility model will not be appropriate for use in cases where malicious hosts behave in very complex ways (either deterministically or stochastically). This is highlighted by Example 9.4, in which malicious hosts attempt to frame a non-malicious host. That is, the possibility that malicious

hosts collude to frame a non-malicious host and, at the same time, the value of CC is misleadingly high, cannot be precluded. However, while the estimate for CC cannot capture all the ways in which a set of malicious hosts may choose to behave, we claim that the credibility evaluation model is still useful in establishing a probabilistic result. Note also that the model should not be used to make predictions in the case of selective or worst-possible malicious host behaviour.

9.4 Impact analysis of malicious behaviour

9.4.1 Procedure

In this section we attempt to analyse the statistical properties of host identification results obtained for a range of possible of behaviour choices.

We first need to define our notion of malicious behaviour choice, as follows.

Definition 9.25. *Suppose that a malicious host $\boldsymbol{c}_j$ is contained in (incident with) a set of k routes $\{\boldsymbol{r}_i, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_k\}$. We define the* vector of behaviour *of $\boldsymbol{c}_j$ to be the binary vector $B_j = \{b_1, b_2, \ldots, b_k\}$, where $b_m = 1$, $1 \leq m \leq k$, if $\boldsymbol{c}_j$ misbehaves for route $\boldsymbol{r}_m$, and $b_m = 0$ otherwise.*

In the rest of this section, we only consider NGT spy agent schemes. We analyse the classification results obtained for all possible malicious host behaviour choices with the use of the following algorithm.

Algorithm 9.26. *Suppose that a set $S = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_n\}$ of hosts is tested using an $\overline{d}$-classifier NGT route design, $d < n$. The following procedure can be used to analyse the impact of random malicious host behaviour.*

Step 1. *Choose a set $\Delta \subset S$ to be the set of malicious hosts, $|\Delta| \leq d$.*

Step 2. *Choose a set of vectors of behaviour* $B = \{B_j : \boldsymbol{c}_j \in \Delta\}$ *and calculate the route design outcome vector. Identify the set* I *of malicious hosts using Corollary 6.18 (i.e. let* I *be the set of hosts that are not included in the union of all routes that give a negative result). Let* $\lambda = |I \cap \Delta|/|\Delta|$*, where* λ *is the classification ratio of successfully identified malicious hosts.*

Step 3. *Repeat Step 2 for all possible choices of* B*. [If each malicious host is visited by* k *agents then there are* $\sigma = 2^{kd}$ *different choices.]*

Step 4. *Repeat steps 2 and 3 for all possible choices of* Δ*. [Suppose that there are* τ *such choices].*

Step 5. *Calculate the* statistical credibility *of the classification algorithm as* $\mathrm{SC} = \phi/(\tau\sigma)$*, where* $\phi = \sum_{\Delta}\sum_{B}\lambda$*.*

9.4.2 Case study

We now provide a simple case study of the use of Algorithm 9.26. In this case study we consider a spy agent route design based on the Fano Plane, shown in Table 6.2, in which there are $v = 7$ routes, $b = 7$ hosts, $k = 3$ routes per host and $r = 3$ hosts per route. We assume that there are exactly two malicious hosts.

We use Algorithm 9.26 to study the impact of random malicious host behaviour.

Step 1. Suppose we choose $\Delta = \{\boldsymbol{c}_1, \boldsymbol{c}_7\}$.

Steps 2 and 3. The two malicious hosts are contained in the following routes: $\boldsymbol{c}_1 = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3\}$ and $\boldsymbol{c}_7 = \{\boldsymbol{r}_3, \boldsymbol{r}_5, \boldsymbol{r}_6\}$. Suppose the vectors of behaviour for the two hosts are $B_1 = (b_1, b_2, b_3)$ and $B_7 = (b_4, b_5, b_6)$,

where $b_1, b_2, b_3, b_4, b_5, b_6$ correspond to the routes $\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3, \boldsymbol{r}_3, \boldsymbol{r}_5, \boldsymbol{r}_6$, respectively. Clearly, there are $2^{kd} = 2^6 = 64$ different ways for choosing B_1 and B_7. That is, there are 64 different ways in which the two hosts can behave.

We show that a host is identified as malicious if and only if all the (three) routes that include this host give a positive result. As discussed in §6.4.3, the Fano Plane is 2-disjunct, and hence the union of any two columns (hosts) does not contain any other column. Thus, a malicious host cannot be included in three routes that test positive unless it misbehaves at least once. [That is, if there are no more than two malicious hosts, then the spy agent scheme based on the Fano Plane is frameproof.] It follows that both hosts $\boldsymbol{c}_1$ and $\boldsymbol{c}_7$ will be identified if and only if $b_1 = b_2 = b_5 = b_6 = 1$ and at least one from b_3, b_4 is also one. That is, both malicious hosts are identified in 3 of the 64 different behavioural cases. Furthermore, only one malicious host, say $\boldsymbol{c}_1$, is identified if and only if $b_1 = b_2 = b_3 = 1$ and at least one from b_4, b_5, b_6 is zero, or $b_1 = b_2 = b_4 = 1$, $b_3 = 0$, and at least one from b_5, b_6 is zero. That is, only $\boldsymbol{c}_1$ is identified in $7 + 3 = 10$ behavioural cases. Similarly, only $\boldsymbol{c}_2$ is identified in a further 10 behavioural cases. If both hosts are identified then $\lambda = 1$. If only one host is identified then $\lambda = 1/2$. If no malicious host is identified then $\lambda = 0$.

Steps 4 and 5. As a result of the symmetry of the Fano plane, any choice of Δ containing two malicious hosts will yield identical results in steps 2 and 3 to those given above. We thus have:

$$\mathrm{SC} = \frac{\phi}{\tau\sigma} = \frac{\sum_{\Delta}\sum_{B}\lambda}{\tau 2^{kd}} = \frac{\sum_{B}\lambda}{2^{kd}} = \frac{3 + 10 + 10}{64} \simeq 0.36\,.$$

We next illustrate the effect of different possible malicious behaviour choices. We group together all the behaviour patterns (vectors) for each defective host, in which zero, one, two or all three spy agents that visit the host yield a positive result. This clearly defines $4 \times 4 = 16$ groups of malicious behaviour patterns for the two malicious hosts. We then aggregate the classification results for all the 16 malicious behaviour scenarios. The results are shown in Figure 9.13.

We make the following observations regarding this figure.

1. We denote the occurrence of any host in the route of an abused agent by OIAA. We also denote the occurrence of any host in three abused agents by OI3AA. We depict OIAA with a circle, and OI3AA with a cross.

2. We use the tuple (NAAH1, NAAH7) to identify the malicious behaviour patterns, where NAAH1 and NAAH7 represent the number of spy agents abused by the malicious hosts $\boldsymbol{c}_1$ and $\boldsymbol{c}_7$, respectively.

3. In the bottom left box (first pattern of malicious behaviour) both $\boldsymbol{c}_1$ and $\boldsymbol{c}_7$ abuse zero agents (NAAH1 = NAAH7 = 0), and thus OIAA is zero for all seven hosts. In the bottom row (NAAH7 = 0) we observe only one OI3AA, when NAAH1 = 3 (bottom right box). In the top right box (where each malicious host abuses all three visiting agents) both malicious hosts are correctly identified. In other scenarios (boxes), we note that a malicious host may or may not be identified, depending on whether the choices of the two malicious hosts result in one malicious host being included in three routes that test positive.

4. Across all the boxes, we observe that no well-behaved host can be framed for any malicious behaviour choice. Also, statistically, when the choice of

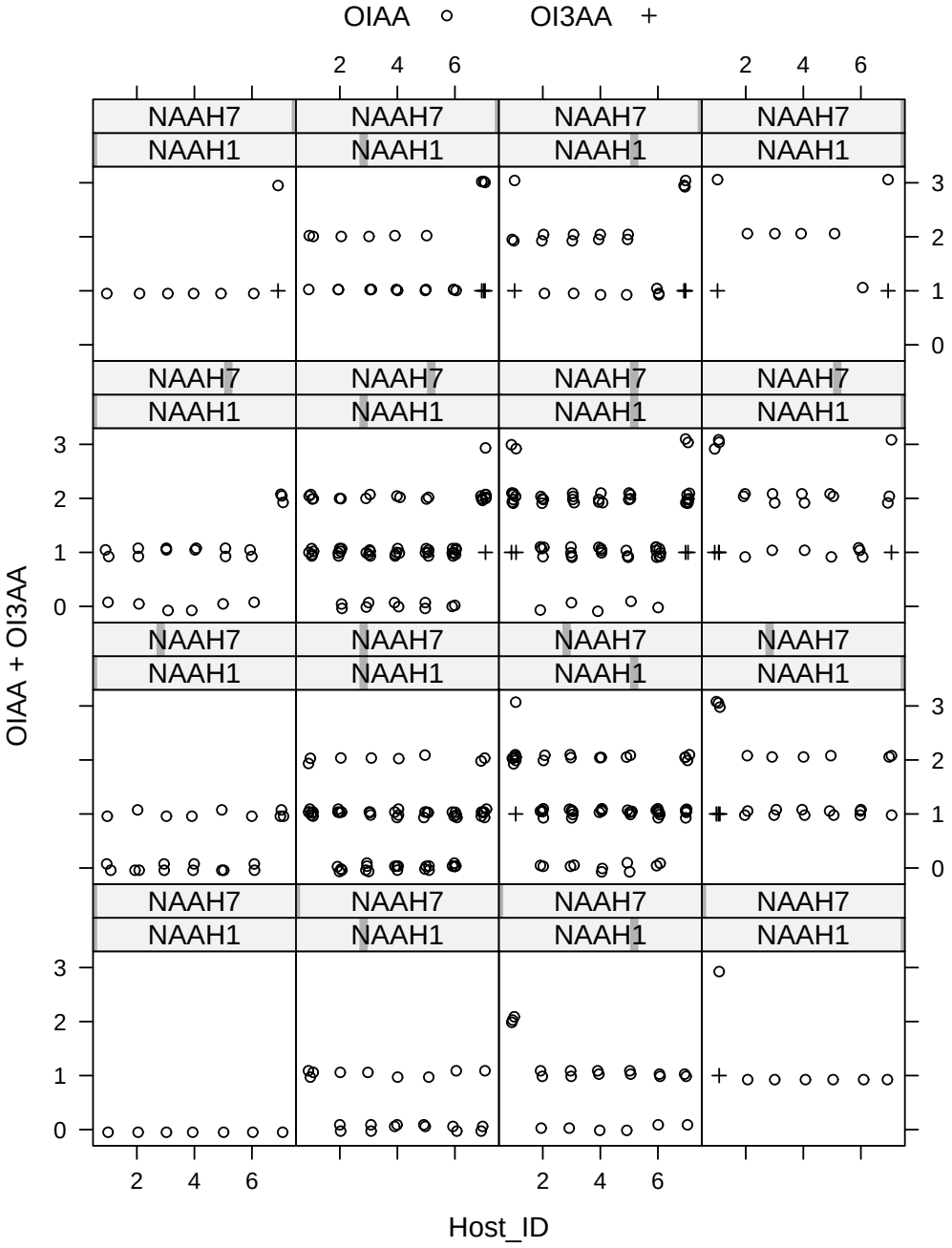


Figure 9.13: Aggregated identification results for different malicious host behaviour patterns of two malicious hosts, based on the route design defined by the Fano Plane.

malicious behaviour is random, a large number of occurrences of a host in malicious routes can be linked with a high likelihood that this host is malicious (although, this might entail statistical errors, as previously discussed).

9.5 Conclusions

In this chapter a model designed to enable the credibility of spy agent results to be evaluated was introduced. This model enables the evaluation of a variety of spy agent routing schemes on the assumption of probabilistic malicious host behaviour. Numerical results based on the model indicate that credibility is maximised when the spy agent route is long enough to mitigate the effects of random or inconsistent malicious host behaviour. However, there are exceptions where mixed results are obtained, especially when malicious hosts only misbehave with a low probability.

The credibility evaluation results can be used to choose appropriate spy agent route designs or algorithms. In this direction, this chapter provides a methodology to evaluate the effectiveness of both NGT designs and SGT algorithms. We have also provided a methodology for analysing the results of all possible behaviour choices that could be made by malicious hosts.

The credibility model is probabilistic, and it can thus exhibit significant statistical errors, especially if malicious hosts collude to try to manipulate the results. This chapter has also shown that, under certain conditions, a careful selection of spy agent routes might provide some protection against framing of hosts.

In future work, spy agent frameproof and error-resilient properties could be further tested and analysed, and more decisive/precise credibility evaluation

models could also be developed. For example, the likelihood of the malicious host collusion scenarios discussed in Chapter 7 could be studied. The probability in erroneous results can be further analysed within the context of error-tolerant route designs.

"I really believe that we don't have to make a trade-off between security and privacy. I think technology gives us the ability to have both."

John Poindexter

10 Spy agent applications

Contents

10.1 Synopsis

This chapter is concerned with the practical application of spy agent systems. In §10.2 we provide a general introduction to the application of spy agents. We then analyse two specific examples of spy agent applications.

- In §10.3 we describe an application of spy agents designed to help address data privacy violation attacks on mobile agents.
- In §10.4 we consider an application of spy agents designed to help address sabotage attacks on mobile agents.

Aspects of the work described in this chapter have been published in [99], and a UK patent has been granted [98].

10.2 Introduction

10.2.1 Setting the scene

In Chapters 6, 7, 8 we introduced and analysed a range of spy agent routing and host classification schemes, and we observed that the choice of scheme depends on the application. In this chapter we consider the application of these schemes in particular practical scenarios.

A spy agent system, as introduced in Chapter 3, combines a spy agent routing scheme with other mobile agent security functions, such as those specified in §2.4, in order to support the required services. This idea is explored further in this section, the remainder of which is structured as follows.

- §10.2.2 outlines the security services that must be provided to support the operation of a spy agent system, and the services that can be provided by such a system;

- §10.2.3 introduces fundamental applicability requirements for spy agent applications; and
- §10.2.4 gives a framework which is subsequently used to specify two examples of practical applications.

10.2.2 Services

10.2.2.1 Required services

The deployment of a set of spy agents can be used to obtain estimates for a variety of security issues, as discussed in §3.4.4. The suitability of a spy agent system for a particular application depends on how well the spy agent requirements defined in Chapter 4 can be met in practice. These requirements imply the need for the following set of services, the provision of which is necessary for the successful application of a spy agent system.

Host security services. These mobile agent system services support secure agent communications and mobility, as described in §4.5 and §4.6. These services help to hold a set of hosts accountable for the secure execution and migration of a visiting mobile agent. Security controls that can be used to provide such security services were discussed in §2.4.4, and a host security architecture was outlined in §2.4.4.2.5.

Mobile code security services. These mobile code execution and migration protection services can be used to detect and/or prevent security attacks on mobile agents. Security controls that can be used to provide such services were discussed in §2.4.3. Schaefer [155] provides an example of such a set of services, in which known security controls are used as building blocks to provide mobile agent e-commerce security services.

In the context of spy agent applications, mobile code security services are required to support threat detection, as described in §4.4.1. Additionally, mobile code security services can help to ensure that malicious hosts process spy agents in the same way as they would other 'standard' mobile agents, as discussed in §4.3.3. The required set of mobile code security services will vary depending on the particular application. Indeed, the applications presented in §10.3 and §10.4 themselves have somewhat different security service requirements.

10.2.2.2 Offered services

Spy agents are designed to identify which hosts attack visiting mobile agents. Providing this service requires a set of spy agents to be disseminated in a manner that satisfies the spy agent dissemblance requirements, given in §4.3, and the attack identification requirements, given in §4.4.2. Spy agent schemes providing such services have been proposed in Chapters 6, 7, and 8. The degree to which the spy agent services are provided will depend on the degree to which the assumptions underlying the spy agent scheme are met.

There are a variety of ways in which a host could attack a mobile agent, as discussed in §2.4.2.1. A particular implementation of a spy agent system might only be capable of detecting certain categories of security attack. In order to test for a range of different security issues, it may therefore be necessary to deploy multiple spy agent systems, each employing a different means of constructing and disseminating spy agents.

Host identification results could be used to provide further services. For example, they could be used as part of the credibility evaluation approach described in Chapter 9, or for risk management, as discussed in §4.6.2.

10.2.3 Applicability

As discussed in §4.4.1, an application of spy agent techniques requires an attack detection mechanism. A tentative list of mobile agent attack detection mechanisms was provided in Table 2.2. We make the following observations regarding the suitability of known attack detection mechanisms for use in spy agent systems.

- Spy agents are likely to be most effective when the impact of an attack on a single mobile agent does not provide sufficient information to identify the host responsible (since malicious hosts are most likely to misbehave in such circumstances).
- Spy agents can be useful in cases where other detection methods are available to help identify malicious hosts; in such a case spy agents could be used to corroborate other assessment results. More generally, a variety of host evaluation techniques could be used in a complementary way.

Example 10.1. Consider a scenario in which agents employ secure chain relation protocols (see §2.4.3.3.7) in order to help identify modification or truncation attacks. Such protocols have potentially large overheads, as discussed in §2.4.3.3.6, which runs counter to the objective of using mobile agents to improve network efficiency. A spy agent system could be used in addition to a secure chain relation protocol in order to corroborate the attack identification results.

Example 10.2. Secure chain protocols, as discussed in Example 10.1, cannot be used to detect DoS and black-box attacks (see §2.4.3.2.2 and §2.4.3.3.2). This can be seen from an attack of the following type.

a. Suppose a malicious host blocks the execution of the agent code in such a way that this action is not recorded by the mobile agent. For example, the host could block the execution of mobile agent logging processes.

b. The host could repetitively execute copies of the agent code until it understands how the agent operates.

c. Using its understanding of the operation of the agent, the host precomputes responses for a variety of possible agent execution calls. For example, suppose that the agent and the host are negotiating the details of a sale, and the negotiation comprises a sequence of agent requests and corresponding host responses. The host might, in this attack scenario, precompute the sequence of responses that is certain to result in the agent accepting an offer that is most advantageous to the host.

d. Finally, the host resumes the original execution of the agent and uses its precomputed sequence of responses to answer the appropriate agent requests without leaving any trace of the black-box attack. For example, the host could adjust its system clock to make sure that the timestamps do not reveal processing delays.

The deployment of detection objects (see §2.4.4.2.3) could be used to detect such black-box attacks. This is discussed further in §10.4.4.

10.2.4 An application framework

We use the following framework to present our specific example applications of spy agent systems.

1. **Outline of operation:** this involves reviewing the objectives of the application, and the assumptions underlying its operation.

2. **Threat detection:** under this heading we describe the means of detecting when a malicious act has been performed on an agent, and justify its suitability for the application concerned.

3. **The scheme:** this involves specifying the system architecture and the spy agent routing scheme.

4. **Alternative applications:** finally, other applications for which similar threat detection mechanisms and spy agent schemes may be suitable are considered.

10.3 Spy agent email honeypots

10.3.1 Outline of operation

Our first example application is the Spy Agent Email Honeypot (SAEH). This involves the use of spy agents to identify mobile data fraudsters and data privacy infringers. Spy agents are used to identify the hosts that are responsible for either mishandling or failing to protect agent PII. This is a 'posterior impact scenario', where the impact of an attack on a mobile agent is realised (potentially a significant time) after the agent visits the malicious host that is responsible for the attack.

Such an application is useful in an environment in which mobile agent hosts hold private mobile agent data covered by privacy protection policies. While thwarting passive security attacks that infringe a privacy agreement is almost impossible if these attacks are deliberately performed by the hosts that hold the data, it is still desirable to identify the hosts that are responsible for such infringements. This security issue was discussed in §2.5.1.3.

The main assumption underlying SAEH is that it is the host's responsibility

to protect the data privacy of a visiting mobile agent. This was argued in §4.2, where we stated that a host is held accountable for any intended or unintended privacy violation resulting from this host accessing, processing, or retaining agent's data. Hence, hosts allowing such infringements can be regarded as malicious and/or untrustworthy.

The main objective of SAEH is to use spy agents to identify malicious hosts that violate the privacy of a visiting agent's email address.

10.3.2 Threat detection

10.3.2.1 The technique

The data privacy violation detection technique involves the use of decoy email addresses. This technique is adapted here from Seigneur and Jensen [158], who describe a proactive privacy protection mechanism. In the Seigneur and Jensen system, individuals subscribe to a number of online providers using, in each case, a unique decoy email address associated with a unique decoy user account. Thus each decoy email address is known by only one provider. If at any later time emails from other providers are sent to this address, then there is a high probability that the email address has been exchanged between the providers involved. This system resembles a honeypot (see §2.5.3.1), in which decoy email addresses are uniquely registered with providers in accordance with a mutual privacy agreement, and the uniqueness of this registration is not known to the providers.

A provider is deemed to be responsible for the reception of any unsolicited email via the associated email address, provided that the 'secrecy' of this unique email address has not been compromised in other ways. A host is held responsible for both attacks in which it abuses the email address itself,

and indirect attacks involving the disclosure of the decoy email address to an unauthorised third party, knowingly or unknowingly (e.g. due to poor storage security). The system enables the exchange of private email addresses between online providers to be tracked, and will thereby provide potentially useful information about the trustworthiness of such providers, such as which providers respect privacy policies and how often private email leakages occur.

The use of honeypot email addresses in the way described by Seigneur et al. [158,159] is limited by the fact that the registration process requires manual input. For the purposes of SAEH we extend and automate this concept by using spy agents and by exposing each decoy email address to a number of hosts, rather than just one. Exposure to multiple hosts is essential for the spy agent concept as it provides a malicious host with an incentive to misbehave, as described in §3.4.3.2. The outcome of a spy agent test is determined by whether or not a host in the agent's route violates the confidentiality of the spy agent's email address. As a result, a positive outcome for a spy agent corresponds to the reception of unsolicited email via the unique email address associated with the agent's route.

10.3.2.2 Applicability

The applicability of the SAEH detection method described above depends on the security characteristics of the email addresses employed in the scheme.

Each spy agent should be equipped with a unique decoy email address. Apart from the entities operating the spy agent system, we assume that a decoy email address will only be known by:

- the subgroup of hosts to which the agent containing it is sent; and
- parties that acquire it illegitimately from other hosts (that themselves

either obtain the address legitimately by receiving an agent containing it or acquire it from another host).

The successful operation of SAEH also depends on the assumption that the email address cannot be found in other networks, and cannot be obtained by any other parties without a host in the agent route ultimately being responsible. In order to make this assumption more tangible, the following secrecy requirements should be met.

- It should not be possible to compromise a decoy email address by a brute-force attack. In particular, each email address must have high entropy.
- A decoy address should not have, or refer to, a 'suspicious' name.

If the above requirements hold, it can be inferred that the reception of any unsolicited email via a decoy email address is evidence that there is at least one untrustworthy host within the destination set of the associated spy agent.

10.3.3 The scheme

10.3.3.1 System architecture

An implementation of SAEH should use a 'standard' structure for mobile agents, as described in §3.4. The spy agent originator creates a set of 'normal' mobile agents and equips them with pseudonymous ID credentials, including decoy email addresses, as discussed in §3.4.2.

10.3.3.2 Route designs

The choice of the routing scheme depends on the assumed behaviour characteristics of malicious hosts. A variety of spy agent routing schemes applying

for different sets of assumptions about malicious host behaviour were given in §6.3.1, §7.2.1 and §8.3.1.

In any application, the choice of the routing scheme depends on when the outcome of a spy agent becomes available. In this case, if an agent's PII (such as an email address) is violated, we assume that the impact of this violation will eventually become evident. As argued in §6.3.2, NGT spy agent schemes are preferable to SGT schemes when the test outcome for a spy agent is only likely to be available after a significant delay. This applies in the SAEH application, as the impact of the misuse of an email address might only become evident some significant time after the related spy agent has commenced its migration.

In addition, NGT schemes allow groups of target hosts (i.e. the hosts in a spy agent route) to be tested at independent times. For example, in a carefully designed spy agent routing scheme, spy agents could be despatched at random times in order to reduce the chance of a malicious host linking them, thereby enhancing the design's subterfuge characteristics (see §4.3.1).

Finally, NGT schemes enable malicious host identification and credibility results to be updated as the test outcomes change over time. For example, if, soon after a spy agent terminates, no violation of the corresponding email address has occurred, then all hosts in the agent route can be classified as well-behaved, albeit with a low evaluation credibility (as analysed in Chapter 9). Over time, the identification result can be updated if the outcome changes following detection of the misuse of an email address. Further background on the subject of PII violations is provided in §2.5.1.

10.3.4 Similar applications

SAEH can be adapted to detect other types of spy agent PII violation, such

as the misuse of credit card information. For example, fraudulent use of credit card details could be detected using spy agents if the spy agent originator cooperated with issuing banks in order to establish decoy sets of credit card details (see Figure 3.4). In such case, the spy agent system would need to cooperate with other trusted services, as discussed in §4.6.1.

10.4 Spy agent shopping honeypots

10.4.1 Outline of operation

Our second application is the Spy Agent Shopping Honeypot (SASH). It involves the use of spy agents to identify the misuse of mobile agent code. Spy agents are used to identify the hosts that are responsible for delaying agent execution as a result of either DoS or black-box attacks (see Example 10.2). This is an 'immediate impact scenario', where the impact of an attack on a mobile agent is realised shortly after the agent visits the malicious host that is responsible for the attack.

Such an application is likely to be useful in an environment where autonomous mobile agents are designed to make purchases on behalf of users, and are protected with code obfuscation techniques (see §2.4.3.2.2).

More specifically, we consider a mobile agent marketplace in which hosts (merchants) are visited by shopping agents (customers). Each agent executes on each visiting host an obfuscated shopping negotiation program, which is designed to protect the confidentiality of the agent's negotiation semantics from the visiting host, at least before the agent migrates elsewhere. While it is hard to prevent a malicious host from analysing the negotiation code's functionality, it is, at least, important to detect such an attack.

The main assumption underlying the operation of SASH is that correctly

operating hosts employ all the security controls necessary to receive, execute, negotiate with, and despatch a shopping agent in a timely and secure manner. A host is held accountable for delaying the execution of an agent regardless of the reason for such a delay, such as negligence or an intention to analyse the agent code.

The main objective of SASH is to use spy agents to identify malicious hosts that attempt to discover the semantics of an agent (see §2.4.2.1). Many schemes have been proposed with the goal of protecting an agent against such attacks, such as encrypted functions (§2.4.3.2.1) and code obfuscation (§2.4.3.2.2); however current techniques are still vulnerable to black-box attacks, as discussed in Example 10.2. Given that all threat prevention mechanisms have practical limitations, the aim of this application of spy agent technology is to provide a threat detection technique that can be used to identify malicious hosts.

10.4.2 Threat detection

10.4.2.1 The technique

The SASH scheme makes the fundamental assumption that the time taken for an agent to complete its migration and negotiation tasks can be used as an indicator of whether or not any of the hosts in its route have attempted to compromise it (e.g. using a DoS or black-box attack). That is, for each spy agent we define a threshold time interval, which will need to be dependent on the length of the agent route, the latency of inter-host communications, and the expected time required to legitimately process the agent. If a spy agent completes its migration in a period of time less than the threshold then the spy agent outcome is deemed to be negative; however, if the agent takes longer

than the threshold (or fails to complete) then the outcome is deemed to be positive.

A shopping spy agent should ideally provide a malicious host with an incentive to mishandle it. For example, the agent could be equipped with high value data enquiries, as discussed in §4.3.4. Also, sending an agent to a significant number of hosts helps to meet the fundamental spy agent incentivisation requirements given in §3.4.3.2.

10.4.2.2 Applicability

The applicability of the SASH detection method described above depends on the predictability of host behaviour. It is reasonable to assume that there are practical scenarios in which all hosts are expected to meet well-defined agent processing deadlines. For example, suppose a host is not able to process a visiting spy agent within the required time period. In this case, a well-behaved host should report the service problem, and allow the agent to migrate to the next host rather than delay execution. In such a scenario, the delay threat detection mechanism is likely to yield reliable outcomes.

The maximum acceptable delay for spy agent migration can be deduced by estimating the time required by a black-box attack to analyse the semantics of the sensitive portions of the agent code. An agent must be designed so that the minimum time required to compromise the code semantics is significantly greater than the maximum time that a host is normally allowed to keep an agent.

10.4.3 The scheme

10.4.3.1 System architecture

SASH spy agents emulate shopping agents equipped with e-commerce negotiation functionality. The use of mobile agents in such applications was briefly discussed in §2.3.4.

In this setting, spy agents could be used to perform comparisons across a range of products and a set of hosts. Spy agents could use a range of different (personalised) criteria to compare products and negotiate deals, covering issues such as price, delivery time, customer service and returns policy [68]. We assume that, given a set of visiting hosts and corresponding set of offers, a spy agent will attempt to negotiate the best possible offer with each host. The encoding of a negotiation strategy could be based on a common ontology such as a *declarative rules language* [162].

The use of distinct product comparison and negotiation strategies by different customers helps to promote merchant differentiation. At the same time, knowledge of agent negotiation semantics could help a malicious host to adapt its offer and maximise its profit (unfairly) at the expense of the customer and, more generally, of the market. This explains why it is desirable for the semantics of the negotiation code in a shopping agent to remain secret. As a result, we assume that agent code is protected using a time-limited black-box technique (see §2.4.3.2.2).

10.4.3.2 Route designs

As with SAEH, the choice of the routing scheme for a SASH application depends on assumptions about the behaviour of malicious hosts. We note that

the assumed lack of significance of the order of spy agents in a route is justified by an assumption that a malicious host is not interested in decoding other hosts' offers; instead, we suppose that the malicious host only wishes to analyse the agent semantics. This assumption is strengthened if a shopping agent migrates through the same set of malicious hosts at least twice (in opposite directions). Such a migration technique will also provide a malicious host with more opportunities (and potential incentives) to misbehave.

The choice between an NGT and a SGT scheme depends on how long it takes for a spy agent to complete its migration. As is generally the case, NGT is preferable when test outcomes are only available after a significant and/or variable delay, while adaptive (sequential) GT is preferable when the test outcomes are produced quickly enough for the tests to be completed in a reasonable time.

10.4.4 Alternative applications

The SASH approach could be adapted to address the detection of other security threats. For example, a GTC-based spy agent scheme (introduced in Chapter 7) could be used to detect inconsistencies in (or corroborate) results obtained from security checks of secure chain protocols, used to detect if malicious hosts have colluded to implement a truncation attack (see §2.4.3.3.7).

10.5 Conclusions

In this chapter we have considered general aspects of the practical application of spy agent techniques. We then investigated two specific examples of possible applications of spy agents.

In the first application, we studied the suitability of spy agent systems

for identifying malicious hosts that infringe the privacy of email addresses of visiting mobile agents. We described a spy agent email honeypot system in which a) the abuse may take place at some time after an agent visits a malicious host, and b) a host may exhibit sophisticated behaviour in order to avoid being linked to the outcome of this behaviour. For example, some malicious hosts might always violate the PII of a visiting spy agent, whereas other malicious hosts might only violate the PII of the spy agent if they identify other malicious hosts in the agent's route. We have described how spy agent systems can be used to help identify the malicious hosts.

In the second application we considered the case where mobile agents migrate between hosts to compare and negotiate quotes. In this case, a malicious host might attempt to gain a competitive advantage by performing a invisible black-box attack that compromises the privacy of the semantics of the negotiation code embedded in the agent. Such an attack can be detected by measuring the overall migration delay. We described a spy agent shopping honeypot system in which spy agents are engineered so that a black-box attack will take a certain amount of time; at the same time, malicious hosts are given an incentive to mount a black-box attack in order to test how the agent responds to a range of different quotes. A spy agent system can use the outcomes computed from the agent migration delays to identify the responsible host(s).

The identified spy agent applications have a number of remaining practical limitations, and therefore need to be developed further before real-world deployment. The necessary development work could benefit from, and contribute to, further advances in other mobile agent protection and threat detection techniques. For example, ongoing research findings in encrypted functions, code

obfuscation, and detection objects could be useful both in providing spy agent applications with appropriate threat detection techniques, and in verifying spy agent results.

"A researcher never really abandons his work; he merely finishes it."

Anonymous

11 Conclusions

Contents

11.1 Synopsis

This chapter concludes this thesis. While specific conclusions were drawn at the end of each main chapter, in this chapter we summarise all the research findings and contributions of this thesis as well as identify possible future research directions.

11.2 Contributions

11.2.1 Spy agents

This thesis is concerned with the design of spy agent systems. Spy agents were introduced in Chapter 3. They can be used to help evaluate the trustworthiness of remote hosts that offer services in mobile code systems, in which programs (software agents) travel from host to host to accomplish their goals.

The main purpose of spy agents is to help identify the origin of detected anomalies, i.e. the hosts responsible for associated attacks. This information could be used in a variety of different ways. For example, mobile agents might avoid identified malicious hosts, and/or law enforcement agencies could use the information to target their investigations. In essence, spy agents help to preemptively protect software agents against the hosts that they visit.

The benefits and significance of spy agents need to be considered within the context of the prior art. As discussed in Chapter 2, mobile code security has for many years been an active and challenging area of research involving two parallel sets of security issues, namely protecting hosts (and other agents) against malicious agents, and protecting agents against malicious hosts. The latter problem is inherently harder to address, since mobile agents are at the mercy of the host which executes them, and ultimately the host can misuse the agent or the agent's data at its discretion. To complicate this problem further, a malicious host might (selectively) misbehave only when it perceives that it can do so without being held accountable. Additionally, a malicious host might (selectively) behave well in order to make a false positive impression.

Spy agents were introduced in this thesis to help address the above problems.

11.2.2 Detailed review

We summarise below the contributions of this thesis on a chapter-by-chapter basis.

- In Chapter 3 we developed the concept of spy agents and we described the core elements of our spy agent system architecture. Such a system uses a set of agents to obtain information reflecting the behaviour of remote hosts, and that can therefore be used to assess their trustworthiness. We described how spy agents could be used to identify a malicious host in cases where common mobile agent security techniques fail through their inability to identify the origin of a detected attack. We further described the principles underlying the development of spy agent systems and their contribution to mobile code security. These principles involve a) spy agents giving malicious hosts incentives to misbehave, and b) methods for collectively evaluating spy agent outcomes. A key issue is that host evaluations depend on the manner in which spy agents are coordinated. We considered a variety of aspects of the selection of both the contents of spy agents (e.g. unique pseudo-ID credentials), and their migration paths (e.g. paths with small correlation and large lengths).
- In Chapter 4 we further developed the spy agent concept by specifying the system requirements. We divided these requirements into four groups, namely spy agent dissemblance requirements (subterfuge, statutory, protection and incentivisation), host evaluation requirements (attack detection and identification, fairness, optimisation), spy agent routing requirements, and trusted services requirements. We also provided the assumptions underlying the design of the spy agent system, covering

both the nature of malicious host behaviour and network security issues such as agent anonymity and host identification.

- In Chapter 5 we analysed a number of spy agent routing architectures adhering to the principles given in Chapter 3 and the system requirements given in Chapter 4. Complementing this analysis we considered how a set of spy agents can be designed to achieve the desired objectives. This analysis provided the basis for the formulation of the spy agent routing problem, as given in the following chapter.

- In Chapter 6 we formulated the spy agent routing and host evaluation problem as a (combinatorial) group testing problem. In group testing theory a (large) population of items containing a small set of defectives is tested in order to identify the defectives. The analogy with spy agents arises from the fundamental spy agent assumption that larger test groups yield more credible results. That is, as discussed in Chapter 3, we assume that the longer the migrating route, the less a malign host is likely to suspect a spying scenario, and the greater the chance that it can cheat without being detected. As a result, a malicious host is more likely to misbehave if it is sent an agent visiting a multiplicity of hosts, and not just one. We examined the properties of non-adaptive group testing algorithms and we showed how these algorithms can be used to obtain 'good' spy agent route sets, i.e. collections of routes that maximise the chance that a malicious host will misbehave and that will provide sufficient information for malicious hosts to be identified. Finally, we proposed the use of a simple class of block designs to construct route sets with the desired properties.

- In Chapter 7 we considered a generalisation of the case studied in Chapter 6 in which some malicious hosts only misbehave collectively, i.e. when they identify other malicious hosts in an agent route. We showed that the spy agent route design problem for such a scenario can be formulated as a non-adaptive complex group testing problem, in which defectives yield a positive test outcome only when other defectives are included in the test. Complex group testing, also known as hypergraph testing, is a relatively young area of research. While in the standard group testing problem we assume that a test result is positive if and only if there is at least one defective item, in this case we assume that a defective consists of a number of items (known as a 'complex'). We analysed known properties of complex group testing, and produced further results. In particular, we developed the concept of complex defectives, and introduced algorithms that can identify individual malicious hosts given an identified set of complex defectives. Finally we proposed the use of a t-designs to construct sets of routes with the desired complex group testing properties. These new results allow us to construct routing designs that are more resilient to complex host behaviour and collusion. Our results could also contribute to the research area of DNA screening group testing problem, which uses hypergraph testing algorithms to identify subsets of molecules that are collectively responsible for the cause of an observed disease.

- In Chapter 8 we considered a different class of spy agent applications, in which malicious behaviour can be detected within a sufficiently short time window to allow the use of sequential group testing algorithms, i.e. algorithms that choose which hosts to test based on the outcomes of

previous tests. We studied the optimisation problem for such a scenario, in which the length of spy agent routes is maximised. We presented and analysed a multi-stage sub-group routing algorithm for this case which we showed to be optimal.

- In Chapter 9 we discussed a scenario in which spy agent evaluations cannot be trusted because of inconsistent results or, more generally, because of inconsistent host behaviour. In this case, the assumptions underlying the host evaluation methods presented in Chapters 6, 7 and 8 do not hold. We discussed the degree to which the results obtained with such methods can be trusted, and we developed a credibility evaluation model. This model can be used to help identify route designs that are less likely to provide erroneous results. For example, we showed how, in certain cases, the results obtained from using route designs containing longer routes are more credible. This occurs because in such cases the longer routes mitigate the effects of random or inconsistent malicious host behaviour. Further, we developed a methodology that analyses the impact of random or inconsistent malicious host behaviour. This provides a methodical way of analysing error-tolerance properties of spy agent route designs.
- In Chapter 10 we discussed possible practical applications of spy agents, and we introduced two specific examples of such applications. The first example involves a spy agent email honeypot system designed to detect email privacy infringements. The second example is a spy agent shopping honeypot system designed to detect black-box attacks. We discussed how these applications might be extended, so that spy agents can be used to help detect the origin of mobile code data privacy and

sabotage attacks. These applications highlight the benefits of the spy agent approach developed in this thesis.

11.3 Future directions for research

We now briefly consider two areas in which further research could be beneficial.

11.3.1 Host testing

As outlined in §11.2.2, in this thesis we have considered a range of scenarios involving specific malicious host behaviour models.

In general, malicious hosts may behave in a wide variety of different ways. As a result, a range of different testing mechanisms, criteria for maximising the incentive to misbehave, and optimum route set constructions are likely to be required.

Future work on spy agent route designs could therefore be approached in the following ways.

- Specific models of malicious host behaviour and attack detection scenarios might need to be defined for individual applications. These could give rise to new problems in spy agent design, depending on the underlying assumptions.
- Conversely, the properties of spy agent route designs of various types could be studied in a more abstract way. Such research might then lead to new spy agent applications.

Elaborating on the above general approaches, we now identify specific possible future research directions.

- The behaviour model of a malicious host might depend on a range of factors, including the degree to which a host is incentivised to misbehave, as discussed in Chapters 4 and 5. Chapters 6, 7 and 8 provide spy agent route designs for specific malicious host behaviour models. In the future it would be useful to define behaviour models based on observed host behaviour. In this direction, it would be interesting to perform practical experiments in which the behaviour of real world malicious hosts is investigated using a series of spy agent evaluations. The objective here would be to obtain a better understanding of how malicious hosts might behave in real scenarios. For example, it might be interesting to investigate:
  a. how the contents of a spy agent can be used to incentivise a malicious host to misbehave;
  b. how the properties of a spy agent route design affect whether or not a malicious host misbehaves; and
  c. to what degree the behaviour of a malicious host depends on the behaviour of other malicious hosts.
- Specific applications and malicious host behaviour models might require particular route design properties. For example:
  a. the NGT design proposed in Chapter 6 is based on the assumption that malicious hosts are more likely to misbehave when the routes of visiting agents are long and are not inter-correlated;
  b. the GTC design proposed in Chapter 7 is based on the assumption that some malicious hosts will only misuse a visiting agent if this agent also visits another malicious host; and

c. the SGT algorithm proposed in Chapter 8 is based on the assumption that the impact of an attack on a spy agent can be detected immediately.

These route designs and algorithms could be enhanced in a variety of different ways, e.g. as follows.

a. It would be interesting to study spy agent routes with error-tolerant and/or frameproof properties, i.e. routes that can be used to identify malicious hosts that behave in unpredictable ways, including those that attempt to frame well-behaved hosts. Such research could draw on and potentially contribute to work on probabilistic or combinatorial group testing.

b. It would also be interesting to study routes with collusion-proof properties, i.e. routes that can be used to identify malicious hosts that are jointly responsible for an attack. This problem becomes harder to address if a joint attack involves more than two malicious hosts. Such research can draw on and potentially contribute to work on complex group testing and DNA testing.

c. Optimality of a spy agent route design can be defined in many ways depending on both application requirements and assumptions regarding the scenario of use. For example, one possible optimisation problem would be to find spy agent route designs that give the highest possible credibility of classification. A model for calculating credibility of classification was proposed in Chapter 9; however, improved models might also be sought.

11.3.2 Mobile code applications

The success of any application of spy agents depends on how well the spy agent requirements are met in the particular application scenario. This in turn depends on which aspect of host trustworthiness is under evaluation, how the impact of host misbehaviour is realised, and how spy agents encourage malicious hosts to misbehave.

The applications given in Chapter 10 could also be extended in many ways, as previously discussed. Further, it would be useful to examine applications in which the nature of a malicious attack can be reflected in the outcome of a spy agent test. In this direction, advances in encrypted functions, code obfuscation, and detection objects might be useful in providing spy agents with the necessary threat detection techniques.

We envisage that, as digital services increase in variety and complexity, software agents and mobile code applications will become increasingly useful. This will provide further motivation for work on future spy agent applications.

A

Code for complex group testing

A.1 Malicious host identification

```
## Calculate spy agent outcome
calcoutput <- function(design=design1,D1=c(1),D2=c(2,3)) {
  complexdef <- calcdefcomplexes(D1,D2)
  output <- c()
  if (length(complexdef)>0)
    for (i in 1:length(design[1,]))
      for (j in 1:length(complexdef[1,]))
        if (contained(complexdef[,j],design[,i])==TRUE)
          output <- c(output, i)
  if (length(output)>0)
    output <- unique(output)
  return(output)
}

## Decoding algorithm to identify complex defectives
finddef2complexes <- function(desT=transpose(design1),
                              output=calcoutput(D1=c(1),D2=c(2,3))) {
  defectives <- c()
  complexes <- combn(length(desT[1,]), 2)
  for (i in 1:length(complexes[1,]))
    if(contained(disj(desT[,complexes[1,i]], desT[,complexes[2,i]]), output)==TRUE)
      defectives <- c(defectives, complexes[,i])
  if (length(defectives >0))
    defectives <- array(defectives,dim=c(2,length(defectives)/2))
  return(defectives)
}
```

A.2 Function library

```
## Create the set of complex defectives
```

```
calcdefcomplexes <- function(D1,D2) {
  All <- c()
  if (length(D1)>0)
    D1 <- sort(unique(D1))
  for (i in 1:length(D1))
    All <- c(All, D1[i], D1[i])
  if (length(D2)>1) {
    D2 <- sort(unique(D2))
    D2 <- combn(D2,2)
    for (j in 1:length(D2[1,]))
      All <- c(All, D2[,j])
  }
  if (length(All)>0)
    All <- array(All,dim=c(2,length(All)/2))
  return(All)
}

## Calculate conjunction
conj <- function(v1, v2) {
  return(unique(sort(c(v1, v2))))
}

## Calculate disjunction
disj <- function(v1, v2) {
  v3 <- sort(c(unique(v1), unique(v2)))
  v4 <- c()
  for (i in 2:length(v3)) {
    if (v3[i]==v3[i-1])
      v4 <- c(v4,v3[i])
  }
  if (length(v4)>0)
    v4 <- unique(sort(v4))
  return(v4)
}

## Check if a set is contained in another
contained <- function(v1, v2) {
  if (length(v2>0))
    if (length(disj(v1,v2))==length(unique(v1)))
      return(TRUE)
  return(FALSE)
}

## Check if a complex defective is contained in another
complexcontained <- function(Tc, Td) {
  tcc <- 0
  for (m in 1:length(Tc[1,]))
    for (n in 1:length(Td[1,]))
      if(contained(Tc[,m], Td[,n])==TRUE)
        tcc <- tcc+1
  return(tcc)
}

## Transpose matrix
transpose <- function(design,h=11,r=30) {
  D <- c()
  for (j in 1:h)
    for (i in 1:length(design[1,]))
      if (length(disj(design[,i],c(j)))>0)
        D <- c(D, i)
  return(array(D, dim=c(r,h)))
}
```

A.3 Resilient rank-2 classification algorithm

```
## Classification algorithm and verification routine
Alg1 <- function(design=design1, h=11, r=30, D1=c(1), D2=c(2,3)) {
  desT <- transpose(design=design,h=h,r=r)
  output <- calcoutput(design=design,D1=D1,D2=D2)
  defs <- finddef2complexes(desT=desT, output=output)
  delta1 <- c()
  dother <- c()
  for (i in 1:length(desT[1,])) {
    Tc <- c()
    l <- length(desT[1,])
    for (j in 1:l)
      if (i!=j)
        Tc <- c(Tc, c(i,j))
    Tc <- array(Tc,dim=c(2,length(Tc)/2))
    tcc <- complexcontained(Tc, defs)
      if (tcc == length(Tc[1,]))
        delta1 <- c(delta1, i)
      else
        dother <- c(dother, i)
  }
  delta2 <- c()
  for (i in 1:length(dother)) {
    for (j in 1:length(defs[1,])) {
      if ((defs[1,j] == dother[i] && !contained(c(defs[2,j]),delta1)) ||
          (defs[2,j] == dother[i] && !contained(c(defs[1,j]),delta1))) {
        delta2 <- c(delta2, dother[i])
        break
      }
    }
  }
  ndl = length(desT[1,])-length(delta1)
  if (length(delta2)==0)
    delta2 <- paste("At most 1 defective host and
                     at least ", ndl-1, " non-defectives", sep="")
  # Collate results
  return(list(output=output, defs=defs, delta1=delta1, delta2=delta2))
}
```

A.4 Disjunctness test

```
## Check (d,e)-disjunct property
checkDdisj <- function(D=transpose(design1),d=2,e=2) {
  h <- length(D[1,])   # 11 hosts
  ld <- length(D[,1])  # 30 routes
  # calculate union of first (e+d) columns
  unions <- D[,(e+1)]
  for (t in (e+2):(e+d))
    unions <- conj(unions, D[,t])
  # calculate first e disjunctions
  disjp <- D[,1]
  for (t in 2:e)
    disjp <- disj(disjp, D[,t])
  #check contained condition
  return(!contained(disjp, unions))
}
```

Bibliography

[1] M. Abe and E. Fujisaki. How to date blind signatures. In *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology*, pages 244–251, London, UK, 1996. Springer-Verlag.

[2] C. Adams and S. Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley, 2nd edition, 2002.

[3] E. Albert, G. Puebla, and M. Hermenegildo. Abstraction-carrying code: a model for mobile code safety. *New Generation Computing*, 26:171–204, 2008.

[4] J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. In *IEEE Symposium on Security and Privacy*, pages 2–11. IEEE Computer Society, 2001.

[5] M. Andreolini, A. Bulgarelli, M. Colajanni, and F. Mazzoni. Honeyspam: Honeypots fighting spam at the source. In *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI '05)*, pages 11–17, Cambridge, MA, USA, July 2005. USENIX Association.

[6] Article 29—Data Protection Working Party. Privacy on the Internet—An Integrated EU Approach to On-line Data Protection. European

Commission WP37 5063/00/EN/FINAL, November 2000. Available at `http://ec.europa.eu/justice/policies/privacy/docs/wpdocs/2000/wp37en.pdf`.

[7] T. Aura and C. Ellison. *Privacy and Accountability in Certificate Systems*. Helsinki University of Technology, 2000.

[8] D. J. Balding, W. J. Bruno, E. Knill, and D. C. Torney. A comparative survey of non-adaptive pooling designs. *Genetic Mapping and DNA Sequencing*, 81:133–154, 1996.

[9] D. J. Balding and D. C. Torney. Optimal pooling designs with error detection. *Journal of Combinatorial Theory, Series A*, 74(1):131–140, 1996.

[10] B. Barak, O. Goldreich, Impagliazzo. R., S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In U. G. Wilhelm, L. Buttyàn, and S. Staamann, editors, *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 1–18, London, UK, 2001. Springer-Verlag.

[11] E. Barillot, B. Lacroix, and D. Cohen. Theoretical analysis of library screening using a n-dimensional pooling strategy. *Nucleic acids research*, 19(22):6241–6247, 1991.

[12] M. Bellare and B. Yee. Forward integrity for secure audit logs. *ACM Transactions on Information and Systems Security*, 23, November 1997.

[13] P. Benassi. TRUSTe: an online privacy seal program. *Communications of the ACM*, 42(2):56–59, February 1999.

[14] S. Berkovits, J. D. Guttman, and V. Swarup. Authentication for mobile agents. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 114–136. Springer Berlin / Heidelberg, 1998.

[15] T. Beth, D. Jungnickel, and H. Lenz. *Design Theory*, volume I. Cambridge University Press, 2nd edition, 1999.

[16] D. Boneh and J. Shaw. Collusion-secure fingerprinting for digital data. *IEEE Transactions on Information Theory*, 44(5):1897–1905, 1998.

[17] N. Borselius. *Multi-agent system security for mobile communication.* PhD thesis, Royal Holloway, University of London, 2003.

[18] N. Borselius. Security for agent systems and mobile agents. In C. J. Mitchell, editor, *Security for Mobility*, chapter 12, pages 287–303. IEE, London, 2004.

[19] N. Borselius, C. J. Mitchell, and A. T. Wilson. On mobile agent based transactions in moderately hostile environments. In B. De Decker, F. Piessens, J. Smits, and E. V. Herreweghen, editors, *Advances in Network and Distributed Systems Security, Proceedings of IFIP TC11 WG11.4 First Annual Working Conference on Network Security*, pages 173–186. Kluwer Academic Publishers, Boston, November 2001.

[20] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment.* Springer-Verlag, 2003.

[21] J. M. Bradshaw. *Software agents.* MIT Press Cambridge, MA, USA, 1997.

[22] P. Braun and W. Rossak. *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*. Morgan Kaufmann, 2005.

[23] B. D. Brunk. Understanding the Privacy Space. *First Monday*, 7(10), 2002.

[24] A. Bürkle, A. Hertel, W. Müller, and M. Wieser. Evaluating the security of mobile agent platforms. *Autonomous Agents and Multi-Agent Systems*, 18:295–311, 2009.

[25] K. A. Bush, W. T. Federer, H. Pesotan, and D. Raghavarao. New combinatorial designs and their application to group testing. *Journal of Statistical Planning and Inference*, 10:335–343, 1984.

[26] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol (SNMP). Technical Report RFC 2401, Internet Engineering Task Force (IETF), May 1990.

[27] C. Castelfranchi. The Role of Trust and Deception in Virtual Societies. *International Journal of Electronic Commerce*, 6(3):55–70, 2002.

[28] P. C. Chan and V. K. Wei. Preemptive distributed intrusion detection using mobile agents. In *Proceedings of the 11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 103–108. IEEE Computer Society, June 2002.

[29] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[30] H. B. Chen, D. Z. Du, and F. K. Hwang. An unexpected meeting of four seemingly unrelated problems: graph testing, DNA complex screening, superimposed codes and secure key distribution. *Journal of Combinatorial Optimization*, 14(2):121–129, 2007.

[31] J. S. L. Cheng and V. K. Wei. Defenses against the Truncation of Computation Results of Free-Roaming Agents. In *Proceedings of the 4th International Conference on Information and Communications Security*, pages 12–24. Springer-Verlag, 2002.

[32] S. G. Choi, A. Elbaz, A. Juels, T. Malkin, and M. Yung. Two-party computing with encrypted data. In *Proceedings of the Advances in Cryptology 13th international conference on Theory and application of cryptology and information security (ASIACRYPT '07)*, pages 298–314, Berlin, Heidelberg, 2007. Springer-Verlag.

[33] B. Chor, A. Fiat, M. Naor, and B. Pinkas. Tracing traitors. *IEEE Transactions on Information Theory*, 46(3):893–910, 2000.

[34] D. N. Chorafas. *Agent technology handbook*. McGraw-Hill, Inc. New York, NY, USA, 1997.

[35] C. J. Colbourn and J. H. Dinitz. *The Handbook of Combinatorial Designs*. CRC Press, 2nd edition, 2006.

[36] A. Corradi, M. Cremonini, R. Montanari, and C. Stefanelli. Mobile agents integrity for electronic commerce applications. *Information Systems*, 24(6):519–533, 1999.

[37] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, Inc. New York, NY, USA, 2006.

[38] H. Cuypers. The mathieu groups and designs. Technical report, Technical University of Eindhoven, 2000. Available at `http://www.win.tue.nl/~hansc/eidmamathieu.pdf`.

[39] D. C. Daniel and K. L. Herbig. Propositions on military deception. *Routledge Journal of Strategic Studies*, 5(1):155–177, 1982.

[40] D. Dasgupta and H. Brian. Mobile security agents for network traffic analysis. In *DARPA Information Survivability Conference and Exposition II*, pages 12–16, Anaheim, California, June 2001. IEEE Computer Society Press.

[41] A. W. Dent and C. J. Mitchell. *User's guide to cryptography and standards*. Artech House, 2005.

[42] Y. Desmedt, R. Safavi-Naini, H. Wang, L. Batten, C. Charnes, and J. Pieprzyk. Broadcast anti-jamming systems. *Computer Networks*, 35(2-3):223–236, 2001.

[43] J. E. Dickerson, J. Juslin, O. Koukousoula, and J. A. Dickerson. Fuzzy intrusion detection. In *Proceedings of the Joint 9th IFSA World Congress and 20th NAFIPS International Conference*, pages 1506–1510. IEEE, July 2001.

[44] R. Dorfman. The detection of defective members of large populations. *Annals of Mathematical Statistics*, 14(4):436–440, 1943.

[45] D. Du and F. Hwang. *Combinatorial Group Testing and Its Applications*. World Scientific, 2nd edition, 2000.

[46] D. Z. Du and F. K. Hwang. *Pooling Designs and Nonadaptive Group Testing*. World Scientific, 2006.

[47] A. D'yachkov, P. Vilenkin, D. Torney, and A. Macula. Families of finite sets in which no intersection of l sets is covered by the union of s others. *Journal of Combinatorial Theory, Series A*, 99(2):195–218, 2002.

[48] A. D'yachkov, P. Villenkin, A. Macula, and D. Torney. On families of subsets where no intersection of l-subsets is covered by the union of s others. *Journal of Combinatorial Theory, Series A*, 99:195–218, 2002.

[49] A. G. D'yachkov, A. J. Macula Jr, and V. V. Rykov. New constructions of superimposed codes. *IEEE Transactions on Information Theory*, 46(1):284–290, 2000.

[50] A. G. D'yachkov and V. V. Rykov. A survey of superimposed code theory. *Problems of Control and Information Theory*, 12(4):1–13, 1983.

[51] M. Dyer, T. Fenner, A. Frieze, and A. Thomason. On key storage in secure networks. *Journal of Cryptology*, 8(4):189–200, 1995.

[52] G. Edjlali, A. Acharya, and V. Chaudhary. History-based access control for mobile code. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 38–48. ACM, NY, USA, 1998.

[53] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO '84 on Advances in cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18, Santa Barbara, California, USA, August 1985. Springer, New York, USA.

[54] P. Erdös, P. Frankl, and Z. Füredi. Families of finite sets in which no set is covered by the union of two others. *Journal of Combinatorial Theory, Series A*, 33:158–166, 1982.

[55] P. Erdös, P. Frankl, and Z. Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel Journal of Mathematics*, 51(1-2):79–89, 1985.

[56] O. Esparza, M. Soriano, J. L. Muñoz, and J. Forné. A protocol for detecting malicious hosts based on limiting the execution time of mobile agents. In *Proceedings of the 8th IEEE International Symposium on Computers and Communications*, pages 251–256. IEEE Computer Society, 2003.

[57] C. F. Fang and G. Radhamani. Security in mobile agent systems. In *Mobile Computing: Concepts, Methodologies, Tools, and Applications*, pages 2600–2613. ICI Global, 2009.

[58] W. Farmer, J. Guttman, and V. Swarup. Security for mobile agents: Authentication and state appraisal. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Computer Security—ESORICS '96*, volume 1146 of *Lecture Notes in Computer Science*, pages 118–130. Springer Berlin / Heidelberg, 1996.

[59] J. Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley, 1998.

[60] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *Proceedings of the 3rd international conference on Information and knowledge management*, CIKM '94, pages 456–463, New York, USA, 1994. ACM.

[61] W. Ford. *Computer communications security: principles, standard protocols and techniques*. Prentice-Hall, New Jersey, NJ, USA, 1994.

[62] E. Gafni, J. Staddon, and Y. L. Yin. Efficient methods for integrating traceability and broadcast encryption. In *Proceedings of the 19th Annual International Conference on Advances in Cryptology*, CRYPTO '99, pages 372–387, London, UK, 1999. Springer-Verlag.

[63] H. Gao, F. K. Hwang, M. T. Thai, W. Wu, and T. Znati. Construction of d(H)-disjunct matrix for group testing in hypergraphs. *Journal of Combinatorial Optimization*, 12(3):297–301, 2006.

[64] J. A. Garay, J. Staddon, and A. Wool. Long-lived broadcast encryption. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 333–352, London, UK, 2000. Springer-Verlag.

[65] C. Gentry. Computing arbitrary functions of encrypted data. *Communications of the ACM*, 53(3):97–105, March 2010.

[66] D. Gollmann. *Computer security*. John Wiley and Sons, Chichester, 1999.

[67] M. Günter and T. Braun. Internet service monitoring with mobile agents. *IEEE Network Magazine*, 16(3):22–29, May / June 2002.

[68] R. Guttman and P. Maes. Agent-mediated integrative negotiation for retail electronic commerce. In *Agent Mediated Electronic Commerce*, volume 1571 of *Lecture Notes in Artificial Intelligence*, pages 70–90. Springer-Verlag, 1999.

[69] C. G. Harrison, D. M. Chess, and A. Kershenbaum. Mobile agents: Are they a good idea? Technical report, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY, March 1995.

[70] M. B. Hasan and P. W. Prasad. A review of security implications and possible solutions for mobile agents in e-commerce. In *Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA '09)*, pages 23–29. IEEE, 2009.

[71] V. Hassler and P. Moore. *Security Fundamentals for E-commerce*. Artech House, 2001.

[72] X. He-qun and F. Deng-guo. Protecting mobile agents' data using trusted computing technology. *Journal of Communication and Computer*, 4(3):44–57, 2007.

[73] R. Hes and J. Borking. Privacy Enhancing Technologies: the path to anonymity (Revised Edition). Registratiekamer, Dutch DPA, A&V-11, 1998.

[74] C. Hewitt. Viewing control structures as patterns of passing messages* 1. *Artificial intelligence*, 8(3):323–364, 1977.

[75] F. Hohl. A Model of Attacks of Malicious Hosts Against Mobile Agents. In *Workshop ion on Object-Oriented Technology*, pages 105–120, London, UK, 1998. Springer-Verlag.

[76] F. Hohl. Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 92–113. Springer Berlin / Heidelberg, 1998.

[77] F. Hohl and K. Rothermel. A protocol preventing blackbox tests of mobile agents. *ITG/VDE Fachtagung Kommunikation in Verteilten Systemen (KiVS '99)*, pages 170–181, 1999.

[78] A. Hotaling. Protecting personally identifiable information on the internet: Notice and consent in the age of behavioral targeting. *CommLaw Conspectus*, 16:529, 2007.

[79] G. Huanmei, M. Xuejun, and Z. Huanguo. A forward integrity and itinerary secrecy protocol for mobile agents. *Wuhan University Journal of Natural Sciences*, 11(6):1727–1730, 2006.

[80] M. N. Huhns and M. P. Singh. *Readings in agents*. Morgan Kaufmann, 1997.

[81] F. K. Hwang. A competitive algorithm to find all defective edges in a graph. *Discrete Applied Mathematics*, 148(3):273–277, 2005.

[82] F. K. Hwang and V. T. Sòs. Non-adaptive hypergeometric group testing. *Studia Sci. Math. Hungar.*, 22:257–263, 1987.

[83] International Organization for Standardization, ISO 7498-2. *Information processing systems—Open systems Interconnection—Basic reference model—Part 2: Security Architecture*. Geneva, Switzerland, 1989.

[84] International Organization for Standardization, ISO/IEC 15408-2:2008. *Information technology—Security techniques—Evaluation Criteria for IT Security—Part 2: Security Functional Components*. Geneva, Switzerland, 2008.

[85] International Organization for Standardization, ISO/IEC 9797-1. *Information technology—Security techniques—Message Authentication Codes (MACs)—Part 1: Mechanisms using block cipher*. Geneva, Switzerland, 1999.

[86] International Organization for Standardization, ISO/IEC 9797-2. *Information technology—Security techniques—Message Authentication Codes (MACs)—Part 2: Mechanisms using a dedicated hash-function*. Geneva, Switzerland, 2002.

[87] International Telecommunication Union (ITU-T), Recommendation X.800 (and ISO/IEC 7498-2). *Data Communication Networks: Open System Interconnection (OSI); Security, Structure and Applications—Security Architecture for Open Systems Interconnection for CCITT Applications*. Geneva, 1991.

[88] International Telecommunications Union (ITU-T), Recommendation X.509. *Information technology—Open Systems Interconnection—The Directory: Public-key and attribute certificate frameworks*, November 2008.

[89] W. Jansen and T. Karygiannis. Mobile agent security. NIST Special Publication 800-19, National Institute of Standards and Technology, August 1999.

[90] T. Jech. *Set Theory*. Springer-Verlag, 3rd edition, 2002.

[91] X. Jiang, J. I. Hong, and J. A. Landay. Approximate Information Flows: Socially-based Modelling of Privacy in Ubiquitous Computing. In G. Borriello and L. Holmquist, editors, *UbiComp 2002: Ubiquitous Computing*, volume 2498 of *Lecture Notes in Computer Science*, pages 176–193. Springer Berlin / Heidelberg, 2002.

[92] H. Jin, K. Liu, F. Xian, and Z. Han. A distributed dynamic self-immunity security architecture. In *Proceedings of the 5th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP '02)*, pages 148–151, Beijing, China, October 2002. IEEE.

[93] Hwang F. K. A method for detecting all defective members in a population by group testing. *Journal of the American Statistical Association*, 67:605–608, 1972.

[94] Hwang F. K., Song T. T., and Du D. Z. Hypergeometric and generalized hypergeometric group testing. *SIAM Journal on Algebraic and Discrete Methods*, 2(4):426–428, 1981.

[95] O. Kachirski and R. Guha. Intrusion detection using mobile agents in wireless ad hoc networks. In *Proceedings of the IEEE Workshop on Knowledge Media Networking (KMN '02)*, pages 153–158. IEEE, 2002.

[96] G. Kalogridis. Network node security analysis method. United Kingdom Patent Office (UKPO), GB2415580, December 2005. Assignee: Toshiba Research Europe Limited.

[97] G. Kalogridis. Network node security analysis using mobile agents. United Kingdom Patent Office (UKPO), GB2428315, January 2007. Assignee: Toshiba Research Europe Limited.

[98] G. Kalogridis. Method for identification of insecure network nodes. United Kingdom Patent Office (UKPO), GB2452555, March 2009. Assignee: Toshiba Research Europe Limited.

[99] G. Kalogridis. Protecting Mobile Code Privacy With Resilient Spy Agent Group Testing. In C. A. Ardagna, S. De Capitani di Vimercati, C. D. Jensen, and R. Küsters, editors, *Proceedings of the Fifth International Workshop on Security and Trust Management (STM '09)*, Electronic Notes in Theoretical Computer Science (ENTCS), pages 58–71, Saint Malo, France, September 2009. Elsevier.

[100] G. Kalogridis and C. J. Mitchell. Using nonadaptive group testing to construct spy agent routes. In S. Jakoubi, S. Tjoa, and E. R. Weippl, editors, *Proceedings of the Third International Conference on Availability, Reliability and Security (ARES '08)*, pages 1013–1019, Barcelona, Spain, March 2008. IEEE Computer Society.

[101] G. Kalogridis, C. J. Mitchell, and G. Clemo. Spy agents: Evaluating trust in remote environments. In Hamid R. Arabnia, editor, *Proceedings of the 2005 International Conference on Security and Management (SAM '05)*, pages 405–411, Las Vegas, Nevada, USA, June 2005. CSREA Press.

[102] Y. Kalyani and C. Adams. Privacy Negotiation using a Mobile Agent. In *Canadian Conference on Electrical and Computer Engineering (CCECE '06)*, pages 628–633. IEEE, May 2006.

[103] G. Karjoth, N. Asokan, and C. Gülcü. Protecting the computation results of free-roaming agents. *Personal and Ubiquitous Computing*, 2(2):92–99, 1998.

[104] N. M. Karnik. *Security in mobile agent systems*. PhD thesis, University of Minnesota, 1998.

[105] W. Kautz and R. Singleton. Nonrandom binary superimposed codes. *IEEE Transactions on Information Theory*, 10(4):363–377, 1964.

[106] N. Krawetz and H. F. Solutions. Anti-honeypot technology. *Security & Privacy Magazine, IEEE*, 2(1):76–79, 2004.

[107] R. Kumar, S. Rajagopalan, and A. Sahai. Coding constructions for blacklisting problems without computational assumptions. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 609–623, London, UK, 1999. Springer-Verlag.

[108] A. D. Lakhani. Deception Techniques Using Honeypots. Master's thesis, Royal Holloway, University of London, 2003.

[109] S. Lei, J. Liu, C. Deng, and J. Xiao. A novel free-roaming mobile agent security protocol against colluded truncation attacks. In *International Conference on Information and Automation (ICIA '08)*, pages 348–353. IEEE Computer Society, 2008.

[110] C. H. Li. A sequential method for screening experimental variables. *Journal of the American Statistics Association*, 57:455–477, 1962.

[111] A. J. Macula. A simple construction of d-disjunct matrices with certain constant weights. *Discrete Mathematics*, 162(1):311–312, 1996.

[112] A. J. Macula, V. V. Rykov, and S. Yekhanin. Trivial two-stage group testing for complexes using almost disjunct matrices. *Discrete Applied Mathematics*, 137(1):97–107, 2004.

[113] S. Maffeis, M. Abadi, C. Fournet, and A. Gordon. Code-carrying authorization. In S. Jajodia and J. Lopez, editors, *Computer Security (ESORICS '08)*, volume 5283 of *Lecture Notes in Computer Science*, pages 563–579. Springer Berlin, 2008.

[114] P. Maggi and R. Sisto. A configurable mobile agent data protection protocol. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, pages 851–858. ACM, 2003.

[115] R. M. Martins, M. R. Chaves, L. Pirmez, and L. F. R. C. Carmo. Mobile agents applications. *Internet Research: Electronic Networking Applications and Policy*, 11(1):49–54, 2001.

[116] C. Meadows. Detecting attacks on mobile agents. In *Proceedings of the DARPA workshop on Foundations for secure mobile code*, pages 64–65, Monterey, USA, 1997. DARPA.

[117] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[118] B. Michael, N. Rowe, M. Auguston, D. Drusinsky, R. Riehle, H. Rothstein, L. Montiero, D. Julian, G. Fragkos, E. Uzuncaova, and T. Wingfield. Intelligent software decoys. *Naval Postgraduate School Research*, 13(1SE):42–44, February (special ed.) 2003.

[119] G. A. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[120] C. J. Mitchell and F. C. Piper. Key storage in secure networks. *Discrete Applied Mathematics*, 21(3):215–228, 1988.

[121] K. D. Mitnick and W. L. Simon. *The art of deception: controlling the human element of security.* John Wiley & Sons, NY, USA, 2003.

[122] T. Moores. Do consumers understand the role of privacy seals in e-commerce? *Communications of the ACM*, 48(3):86–91, March 2005.

[123] National Institute of Standards and Technology (NIST). *Federal Information Processing Standards: Advance Encryption Standard (AES).* Gaithersburg, MD, USA, November 2001.

[124] R. Neches, R. Fikes, T. W. Finin, T. R. Gruber, R. Patil, T. E. Senator, and W. R. Swartout. Enabling Technology for Knowledge Sharing. *AI Magazine*, 12(3):36–56, 1991.

[125] G. C. Necula. Proof-Carrying Code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '97)*, pages 106–119, New York, USA, 1997. ACM.

[126] G. C. Necula and P. L. Safe. Untrusted agents using proof-carrying code. In *Lecture Notes in Computer Science*, volume 1419. Springer-Verlag, 1998.

[127] S. K. Ng and K. W. Cheung. Protecting mobile agents against malicious hosts by intention spreading. In H. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '99)*, volume II, pages 725–729. IEEE, 1999.

[128] H. Q. Ngo and D. Z. Du. A survey on combinatorial group testing algorithms with applications to DNA library screening. In *Discrete Mathematical Problems with Medical Applications*, volume 55 of *DIMACS Discrete Math. Theoret. Comput. Sci.*, pages 171–182, 2000.

[129] H. Q. Ngo and D. Z. Du. New constructions of non-adaptive and error-tolerance pooling designs. *Discrete Math.*, 243(1-3):161–170, 2002.

[130] S. Oaks. *Java Security*. O'Reilly Media, Inc., 2nd edition, 2001.

[131] OECD. *Privacy Online: OECD Guidance on Policy and Practice*. OECD Publishing, November 2003.

[132] OECD. *OECD Glossary of Statistical Terms*. OECD Publishing, September 2008.

[133] National Institute of Standards and Technology (NIST). *Federal information processing standards: Secure Hash Standard*. Gaithersburg, MD, USA, 2002.

[134] J. J. Ordille. When agents roam, who can you trust? In *Proceedings of the 1st Annual Conference on Emerging Technologies and Applications in Communications*, pages 188–194. IEEE, May 1996.

[135] R. Ostrovsky and W. Skeith. Algebraic lower bounds for computing on encrypted data. Cryptology ePrint Archive, Report 2007/064, 2007.

[136] J. K. Ousterhout, J. Y. Levy, and B. B. Welch. The Safe-Tcl Security Model. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 217–234. Springer Berlin / Heidelberg, 1998.

[137] R. S. Patil, R. E. Fikes, P. F. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. *The DARPA knowledge sharing effort: progress report*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[138] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management, August 2010. v0.34. Latest version available from `http://dud.inf.tu-dresden.de/Anon_Terminology.shtml`.

[139] N. Provos. A virtual honeypot framework. In *Proceedings of the 13th USENIX Security Symposium*, pages 1–14, San Diego, CA, USA, August 2004. USENIX Association.

[140] L. Rasmusson, A. Rasmusson, and S. Janson. Using Agents to Secure the Internet Marketplace: Reactive Security and Social Control. In *Proceedings of the 2nd International Conference on the Practical Application*

of Intelligent Agents and Multi-Agent Technology, London, UK, 1997. Springer-Verlag.

[141] J. Reagle and L. F. Cranor. The platform for privacy preferences. *Communications of the ACM*, 42(2):48–55, February 1999.

[142] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected areas in Communications*, 16(4):482–494, 1998.

[143] H. Reiser and G. Vogt. Security Requirements for Management Systems using Mobile Agents. In S. Tohme and M. Ulema, editors, *Proceedings of the Fifth IEEE Symposium on Computers & Communications*, pages 160–165, Washington, DC, USA, 2000. IEEE Computer Society.

[144] M. K. Reiter and A. D. Rubin. Anonymous Web transactions with Crowds. *Communications of the ACM*, 42(2):32–48, February 1999.

[145] J. Riordan and B. Schneier. Environmental key generation towards clueless agents. In G. Vigna, editor, *Mobile Agents and Security*, pages 15–24, London, UK, 1998. Springer-Verlag.

[146] R. L. Rivest. The RC4 Encryption Algorithm. Technical report, RSA Data Security, Redwood City, CA, USA, 1992.

[147] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[148] V. Roth. Secure recording of itineraries through cooperating agents. In *Object-Oriented Technology (ECOOP '98)*, volume 1543 of *Lecture Notes in Computer Science*, pages 297–298. Springer-Verlag, 1998.

[149] V. Roth. On the robustness of some cryptographic protocols for mobile agent protection. *Lecture Notes in Computer Science*, 2240:1–14, 2001.

[150] V. Roth. Obstacles to the adoption of mobile agents. In *Proceedings of the IEEE International Conference on Mobile Data Management*, pages 296–297. IEEE, August 2004.

[151] R. Safavi-Naini and H. Wang. New results on multi-receiver authentication codes. In *Advances in Cryptology (Eurocrypt '98)*, volume 1403 of *Lecture Notes in Computer Science*, pages 527–541. Springer, 1998.

[152] T. Sander and C. Tschudin. Towards mobile cryptography. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 215–224, Oakland, CA, May 1998. IEEE.

[153] T. Sander and C. F. Tschudin. On software protection via function hiding. In *Information Hiding*, volume 1525 of *Lecture Notes in Computer Science*, pages 111–123. Springer Berlin / Heidelberg, 1998.

[154] T. Sander and C. F. Tschudin. Protecting Mobile Agents Against Malicious Hosts. In G. Vigna, editor, *Mobile Agents and Security*, pages 44–60, London, UK, 1998. Springer-Verlag.

[155] I. Schaefer. Secure Mobile Multiagent Systems In Virtual Marketplaces: A Case Study on Comparison Shopping. Technical Report RR-02-02, Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI GmbH, March 2002.

[156] M. Schillo, P. Funk, and M. Rovatsos. Using Trust for Detecting Deceitful Agents in Artificial Societies. *Applied Artificial Intelligence*, 14(8):825–848, 2000.

[157] F. B. Schneider. Towards fault-tolerant and secure agentry. In M. Mavronicolas and P. Tsigas, editors, *Distributed Algorithms*, volume 1320 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin / Heidelberg, 1997.

[158] J. M. Seigneur and C. D. Jensen. Privacy recovery with disposable email addresses. *IEEE Security and Privacy*, 1(6):35–39, 2003.

[159] J. M. Seigneur, A. Lambert, P. Argyroudis, and C. D. Jensen. PR3 email honeypot. Technical Report TCD-CS-2003-39, Department of Computer Science, University of Dublin, 2003. Available at `https://www.cs.tcd.ie/publications/tech-reports/reports.03/TCD-CS-2003-39.pdf`.

[160] S. W. Shah, P. Nixon, R. I. Ferguson, S. R. Hassnain, M. N. Arbab, and L. Khan. Securing Java-Based Mobile Agents through Byte Code Obfuscation Techniques. In *Proceedings of the IEEE Multitopic Conference (INMIC '06)*, pages 305–308. IEEE, December 2006.

[161] V. Shoup. Practical threshold signatures. In B. Preneel, editor, *Advances in Cryptology (Eurocrypt '00)*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer Berlin / Heidelberg, 2000.

[162] T. Skylogiannis, G. Antoniou, N. Bassiliades, G. Governatori, and A. Bikakis. DR-NEGOTIATE—A system for automated agent negotiation with defeasible logic-based strategies. *Data & Knowledge Engineering*, 63(2):362–380, 2007.

[163] M. Sobel and P. A. Groll. Binomial group-testing with an unknown proportion of defectives. *Technometrics*, 8(4):631–656, 1966.

[164] D. J. Solove, M. Rotenberg, and P. M. Schwartz. Information Privacy 900—European Union Data Protection Directive of 1995, Directive 95/46/EC. Available at `http://ec.europa.eu/justice_home/fsj/privacy/law/index_en.htm`.

[165] E. H. Spafford and D. Zamboni. Intrusion detection using autonomous agents. *Computer Networks*, 34(4):547–570, 2000.

[166] L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley Professional, 2002.

[167] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. Technical Report RFC 2663, Internet Engineering Task Force (IETF), August 1999.

[168] R. P. Srivastava and T. J. Mock. Evidential reasoning for WebTrust assurance services. *Journal of Management Information Systems*, 10(3):11–32, 1999.

[169] J. N. Staddon, D. R. Stinson, and R. Wei. Combinatorial properties of frameproof and traceability codes. *IEEE Transactions on Information Theory*, 47(3):1042–1049, 2001.

[170] W. Stallings. *Network security essentials: applications and standards*. Prentice Hall, 2007.

[171] I. Stengel, K. P. Fischer, U. Bleimann, and J. Stynes. Mitigating the mobile agent malicious host problem by using communication patterns. *Information Management and Computer Security*, 13(3):203–211, 2005.

[172] D. R. Stinson, T. Van Trung, and R. Wei. Secure frameproof codes, key distribution patterns, group testing algorithms and related structures. *Journal of Statistical Planning and Inference*, 86(2):595–617, 2000.

[173] D .R. Stinson and R. Wei. Combinatorial properties and constructions of traceability schemes and frameproof codes. *SIAM Journal on Discrete Mathematics*, 11(1):41–53, 1998.

[174] D. R. Stinson and R. Wei. Key preassigned traceability schemes for broadcast encryption. In *Proceedings of the Selected Areas in Cryptography*, SAC '98, pages 144–156, London, UK, 1999. Springer-Verlag.

[175] D. R. Stinson and R. Wei. Generalized cover-free families. *Discrete Mathematics*, 279(1-3):463–477, 2004.

[176] K. Sycara and D. Zeng. Coordination of Multiple Intelligent Software Agents. *International Journal of Cooperative Information Systems*, 5:181–212, 1996.

[177] H. K. Tan and L. Moreau. Extending execution tracing for mobile code security. In K. Fischer and D. Hutter, editors, *Procedings of 2nd International Workshop on Security of Mobile MultiAgent Systems (SEMAS '02)*, pages 51–59, Bologna, Italy, June 2002. Deutsches Forschungszentrum fur Kunstliche Intelligenz, DFKI Saarbrucken.

[178] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2002.

[179] J. Tao, L. Ji-ren, and Q. Yang. The research on dynamic self-adaptive network security model based on mobile agent. In *Proceedings of 36th International Conference on Technology of Object-Oriented Languages and Systems*, pages 134–139, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[180] D. C. Torney. Sets pooling designs. *Annals of Combinatorics*, 3(1):95–101, 1999.

[181] E. Triesch. A group testing problem for hypergraphs of bounded rank. *Discrete Applied Mathematics*, 66(2):185–188, 1996.

[182] UK Act of Parliament. *Data Protection Act*. The Stationery Office Limited, July 1998.

[183] V. Varadharajan, N. Kumar, and Y. Mu. An approach to designing security model for mobile agent based systems. In *Global Telecommunications Conference (GLOBECOM '98)*, pages 1600–1606, Sydney, 1998. IEEE.

[184] G. Vigna. Protecting mobile agents through tracing. In *Proceedings of the 3rd Workshop on Mobile Object Systems*. Jyvälskylä, Finland, June 1997.

[185] G. Vigna. Cryptographic traces for mobile agents. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 137–153. Springer-Verlag, Berlin, 1998.

[186] G. Vigna. *Mobile Code Technologies, Paradigms, and Applications*. PhD thesis, Politecnico di Milano, 1998.

[187] J. Vitek and G. Castagna. Seal: A Framework for Secure Mobile Computations. In *Workshop on Internet Programming Languages*, pages 47–77, London, UK, 1999. Springer-Verlag.

[188] H. Vogler, A. Spriestersbach, and M. L. Moschgath. Protecting competitive negotiation of mobile agents. In *Proceedings of the IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '99)*, pages 145–150, Cape Town, South Africa, December 1999. IEEE.

[189] C. Wang, J. Hill, J. Knight, and J. Davidson. Software tamper resistance: Obstructing static analysis of programs. Technical Report CS-2000-12, Department of Computer Science, University of Virginia, Charlottesville, VA, USA, 2000.

[190] R. Wei. On cover-free families, 2006. Preprint available at `http://ccc.cs.lakeheadu.ca/psfiles/CFF.ps`.

[191] G. Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT Press, 2000.

[192] L. Weiwei, H. Zhen, and W. Qinglong. An Approach to the Sensitive Information Protection for Mobile Code. In *Proceedings of the The First International Symposium on Data, Privacy, and E-Commerce*, pages 289–297, Washington DC, USA, 2007. IEEE Computer Society.

[193] U. G. Wilhelm. Cryptographically protected objects. Technical report, Ecole Polytechnique Fédérale de Lausanne, Switzerland, 1997.

[194] U. G. Wilhelm, S. Staamann, and L. Buttyan. On the problem of trust in mobile agent systems. In *Symposium on Network and Distributed System Security*, pages 114–124. ACM, 1998.

[195] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu. Negotiating trust in the web. *IEEE Journal of Internet Computing*, 6(6):30–37, 2002.

[196] M. Withall, I. Phillips, and D. Parish. Network visualisation: a review. *IET Communications*, 1(3):365–372, 2007.

[197] J. Wolf. Born again group testing: Multiaccess communications. *IEEE Transactions on Information Theory*, 31(2):185–191, 1985.

[198] D. Xu, L. Harn, M. Narasimhan, and J. Luo. An Improved Free-Roaming Mobile Agent Security Protocol against Colluded Truncation Attacks. In *Proceedings of the 30th Annual International Computer Software and Applications Conference*, volume 2. IEEE Computer Society, 2006.

[199] B. S. Yee. *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, 1994.

[200] B. S. Yee. A sanctuary for mobile agents. In J. Vitek and C. D. Jensen, editors, *Secure Internet programming*, pages 261–273. Springer-Verlag, London, UK, 1999.

[201] Y. Zhang and W. Lee. Intrusion detection in wireless ad-hoc networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 275–283. IEEE, 2000.

[202] J. Zhou, J. A. Onieva, and J. Lopez. Analysis of a free roaming agent result-truncation defense scheme. In *Proceedings of the IEEE International Conference on e-Commerce Technology*, pages 221–226, Washington, DC, USA, 2004. IEEE Computer Society.