MAgSeM: A Multi-agent based Security Model for Secure Cyber Services

By

Rossilawati Sulaiman
B Sc. (Computer Science)
M Sc. (Computer Science)

The Faculty of Information Sciences and Engineering
University of Canberra ACT 2601 Australia

PhD Thesis

A Thesis submitted in fulfilment of the requirements of the Degree of Doctor of Philosophy (Information Sciences and Engineering)

(August 6, 2010)
Abstract

Ever since people started to become aware of the value of information, they have been conscious about the underlying security issues. Reliance on the Internet as a medium of communication to exchange and share information has become prevalent. Electronic health (or e-health) uses the Internet to enhance healthcare service deliveries. Current practices in e-health involve applications that support online communication such as videoconferencing sessions, electronic mails, web-based applications, and also software applications used with mobile devices. Remote patients and medical staff communicate and exchange messages regarding e-health issues such as patient consultations, diagnosis, and appointment requests. Medical staff can also monitor patients remotely.

However, while the Internet greatly facilitates and enhances these services, significant threats also come in parallel. Network attacks, information privacy/sensitivity breaches, and malicious software, which involve programs that are purposely created to perform illegal operations (such as viruses and worms) on a computer system, are common types of threats to Internet communication. These threats can cause severe damage to computer systems as well as to the information. The information might be stolen or modified or even eavesdropped on and all these may cause undesirable consequences. Therefore, it is imperative that online communication is secure.

Using these problems as motivation, we proposed a security framework, which caters for the security needs for online communication between two nodes which may have similar or dissimilar communicating environments. We introduce a Multilayer Communication approach (MLC) that improves efficiency, security, and robustness by classifying communication between different categories of users into five different layers based on requirements: Layer 1 to Layer 5, namely Extremely Sensitive, Highly Sensitive, Medium Sensitive, Low Sensitive and No Sensitive Data. This classification is based on the different sensitivity of the information being exchanged during communication. For example, Extremely Sensitive communication involves exchanging extremely sensitive information.

E-health security was the motivating problem. The various categories of users in e-health are identified, so that we can determine the sensitivity level of the information that may be exchanged between the users. Then the layer of the communication (Layer 1 to Layer 5) is
determined, to find the most suitable security mechanisms that should be applied to the communication. Data security and/or channel security are provided at each layer depending on the sensitivity of the data. Highest security mechanisms are applied to the extremely sensitive information, while low security mechanisms are applied to the low sensitive information. Cryptography protocols such as encryption/decryption, digital signature, and hash function are used and applied on the data, while secure socket layer (SSL/TLS) is used to secure the communication channel.

A novel multi-agent system architecture is developed to cater for the security processes to secure the communications at the various levels conceptualised at each layer. The agents are skilled with the knowledge to cater for the relevant security processes. Mobile agents are used as supporting tools to carry sensitive data from the Sender’s side to the Recipient’s side. Cryptographic protocols are used to secure the data as well as the mobile agent code, which provide mechanisms to verify the authenticity, confidentiality and the integrity of data, and decipher the data and code received by the recipient nodes. Here, appropriate MLC is identified and used real time when selecting the security protocols.

Experiments have been conducted on the proof-of-concept and tested using the Jade platform. The performance of each layer in MLC is investigated and we concluded that Layer 1 has the highest overhead compared to the other layers due to the highest security overheads applied in this layer based on the level of security requirements. Results also showed that agents incur a higher cost compared to the traditional method but these costs are largely due to communication requirements. However, the proposed architecture gives a much better control on security to the initiator for the end-to-end channels. The recipient nodes do assume any security control unlike most existing communicating nodes on networks. The proposed novel model contributes significantly to research in security for a class of problems that have distributed IT solutions over data networks. The e-Health problem was the motivating problem for the research. Its characteristic needs were adequately addressed by the model with increased robustness in security and improvement in efficiency.

*Keywords:* security, mobile agent, multi-agent system, e-health.
Acknowledgements

I would like to take this opportunity to express my sincere gratitude to my supervisors Prof Dr Dharmendra Sharma, Dr Wanli Ma, and Dr Dat Tran for their ongoing support and encouragement, kindness and guidance throughout my four years of research in the university.

Many thanks to all the ISE staff at the University of Canberra, who were involved whether directly or indirectly, and provided essential environment for the completion of this study; especially to Serena Chong, Coral Suthern, Hanh Huynh, Xu Huang, Claire Dunstan, and Jason Weber. My thanks also go to Fariba Shadabi and Girija Chetty for your ongoing encouragement.

Great thanks to other researchers in Room 11B-31, Kavitha Gurusamy, Sisira Adikari, Len Bui, Trung Le, and Dat Huynh; your hard work and support always inspire me to survive through these years. Many thanks for the friendship, and always make me feel at home.

Also, I wish to express my deepest thanks to my husband, Fadzli, who was there for me through thick and thin and continuously encouraged me; and to my lovely little son Amin, for giving me all the smiles and courage that I never thought I have to finish this thesis. Without your support this thesis would not have been possible. Also, my warmest thanks go to my family back home for their ongoing support and prayers.

Last, but not least, it is my pleasure to acknowledge the financial support for this thesis: National University of Malaysia and the Ministry of Higher Education of Malaysia.
### Table of Content

**Abstract** ........................................................................................................................................ v

**Acknowledgements** ................................................................................................................... vii

**Table of Content** ......................................................................................................................... ix

**List of Figures** .......................................................................................................................... xiii

**List of Tables** ........................................................................................................................... xvii

**List of Acronyms** ...................................................................................................................... xxi

**CHAPTER 1 INTRODUCTION** ........................................................................................... 1

1.1 Motivations for the Research ....................................................................................... 1

1.2 Research Questions ..................................................................................................... 5

1.3 Proposed Research ....................................................................................................... 5

1.4 Contributions ............................................................................................................. 6

1.5 Thesis Organization ..................................................................................................... 7

**CHAPTER 2 RECENT ADVANCES: SECURITY CONTEXT** .......................................... 9

2.1 Introduction .............................................................................................................. 9

2.2 Computer Networks and Security................................................................................ 9

2.2.1 Open System Interconnection (OSI) Model ...................................................... 10

2.2.2 Network Security Attacks .................................................................................. 13

2.2.3 Security Architecture and the OSI Model ......................................................... 15

2.3 A Review of Existing Security Systems ....................................................................... 18

2.3.1 Firewall .............................................................................................................. 18

2.3.2 Intrusion Detection Systems .............................................................................. 21

2.3.3 Cryptography ..................................................................................................... 22

2.3.4 Cryptography Systems ....................................................................................... 26

2.4 Security Applications ................................................................................................ 33

2.5 Agents and Security ................................................................................................... 37

2.5.1 Agents ................................................................................................................ 38

2.5.2 Agent Characteristics ......................................................................................... 38

2.5.3 Multi-agent Systems and characteristics ........................................................... 40

2.5.4 MAS and Security ............................................................................................. 42

2.6 Gaps in Security: Motivations ................................................................................... 54
CHAPTER 5  PROPOSED MULTI-AGENT SECURITY MODEL

5.1  Introduction ............................................................................................................. 129
5.2  Identifying Agents Goals against Organizational Structure .................................. 130
  5.2.1  Organizing the Agents ..................................................................................... 132
5.3  MAgSeM Architecture ............................................................................................ 134
  5.3.1  Communication Layers ..................................................................................... 136
  5.3.2  MLC Specification .......................................................................................... 138
  5.3.3  Control over Data by Sender ........................................................................... 139
  5.3.4  Security Mechanism in MAgSeM ................................................................... 141
  5.3.5  Advantages of the Control Mechanism ........................................................... 144
5.4  MAgSeM Communication Architecture ................................................................. 145
  5.4.1  Certificates and Keys ....................................................................................... 145
  5.4.2  Message Format ............................................................................................... 146
  5.4.3  Different Agents Actions ................................................................................. 146
5.5  SUMMARY ............................................................................................................ 151

CHAPTER 6  THE MAgSeM SYSTEM

6.1  Introduction ............................................................................................................. 153
6.2  Supporting Tools ..................................................................................................... 153
  6.2.1  Java Agent Development Environment (JADE) ........................................... 154
  6.2.2  Tools for Mobile Devices Development ......................................................... 160
  6.2.3  Cryptographic library: Bouncy Castle ............................................................. 164
6.3  MAgSeM-based System Implementation ................................................................. 164
  6.3.1  Agent Interactions: Wired System ................................................................... 165
  6.3.2  Agent Interactions: Wireless System ............................................................... 167
  6.3.3  Agent Base Classes Implementation ............................................................... 169
6.4  Socket-based System Implementation .................................................................... 193
  6.4.1  Client-side Implementation ............................................................................. 193
  6.4.2  Server-side Implementation ............................................................................. 195
  6.4.3  Communication Link ....................................................................................... 196
6.5  Summary .................................................................................................................. 199
List of Figures

Figure 2-1: The seven layer of OSI model ..............................................................10
Figure 2-2: The TCP/IP and OSI layer .................................................................12
Figure 2-3: Firewall between two networks .........................................................19
Figure 2-4: Demilitarized zone .............................................................................20
Figure 2-5: IPSec Transport Mode .......................................................................29
Figure 2-6: IPSec Tunnel Mode .............................................................................30
Figure 2-7: Agency properties ...............................................................................38
Figure 2-8: Certification and information delivery phase ....................................45
Figure 2-9: Monitoring phase ................................................................................46
Figure 2-10: Agent platform with SCA ...............................................................48
Figure 2-11: RETSINA architecture .................................................................49
Figure 2-12: Self-protected mobile agent mechanism .......................................50
Figure 2-13: SECMAp architecture .................................................................51
Figure 2-14: Example of a company network ....................................................58
Figure 2-15: SSH algorithms configuration for Windows Server ....................60
Figure 2-16: IPSec setting on Window XP Professional .....................................62
Figure 2-17: Examples of multilayered structure ..............................................65
Figure 3-1: Different types of communications in a hospital organization ....76
Figure 3-2: Data security between two points ..................................................86
Figure 4-1: Control Topology ............................................................................114
Figure 4-2: Example of FIPA-ACL message ....................................................119
Figure 4-3: Example of KQML message ............................................................120
Figure 5-1: AND/OR graph for the agent’s actions ........................................131
Figure 5-2: Organizing the agents in the layered architecture .........................132
Figure 5-3: Proposed MAgSeM .......................................................................134
Figure 5-4: An example of MLC specifications ...............................................138
Figure 5-5: Maintaining Control over the Data ................................................140
Figure 5-6: Agent Communication in MAgSeM ..............................................146
Figure 6-1: Platform and Containers .................................................................155
Figure 6-2: JADE-Leap runtime environment

Figure 6-3: Examples of sending an ACL message

Figure 6-4: Examples of receiving an ACL message

Figure 6-5: J2ME protocol stack

Figure 6-6: Jade Environment for Mobile Device

Figure 6-7: Communication protocols performed on the Wired system

Figure 6-8: Communication protocols performed on Wireless to Wired systems

Figure 6-9: Interface for user authentication

Figure 6-10: Interface for message composition

Figure 6-11: List of recipients

Figure 6-12: Interfaces for the MAgSeM-based Wireless system

Figure 6-13: MTA sending a request to every recipient’s CLA

Figure 6-14: Determining com_layer

Figure 6-15: Implementation of MLC Specification

Figure 6-16: cA sends partial results to MTA

Figure 6-17: MTA’s actions

Figure 6-18: Generate symmetric keys for Wired system

Figure 6-19: Generate a symmetric key for the Wireless system

Figure 6-20: Obtaining public and private key

Figure 6-21: Public key extraction

Figure 6-22: Key serializations

Figure 6-23: Symmetric encryption for the Wired system

Figure 6-24: Asymmetric encryption for the Wired system

Figure 6-25: Symmetric encryption for the Wireless system

Figure 6-26: Asymmetric encryption for the Wireless system

Figure 6-27: Binary encoding for the Wired system

Figure 6-28: Binary decoding for the Wired system

Figure 6-29: Binary encoding for the Wireless system

Figure 6-30: Binary decoding for the Wireless system

Figure 6-31: Generate key pairs

Figure 6-32: Signing with a private key
Figure 6-33: Verifying signature with a public key ................................................................. 187
Figure 6-34: Generate message digest .................................................................................... 188
Figure 6-35: Recompute and verify message digest ............................................................... 189
Figure 6-36: Symmetric decryption for the Wired system .................................................... 190
Figure 6-37: Symmetric decryption for the Wireless system ................................................ 190
Figure 6-38: Asymmetric decryption for Wired system ......................................................... 191
Figure 6-39: Asymmetric decryption for Wireless system .................................................... 191
Figure 6-40: Example of Jade.conf ....................................................................................... 192
Figure 6-41: Creating a client-side socket ............................................................................. 193
Figure 6-42: Creating a client-side SSL-based socket .......................................................... 193
Figure 6-43: Creating a client-side socket ............................................................................. 194
Figure 6-44: Sending message ............................................................................................. 194
Figure 6-45: Receiving messages ......................................................................................... 195
Figure 6-46: Creating a server-side socket .......................................................................... 195
Figure 6-47: Creating a server-side SSL-based socket ........................................................ 196
Figure 6-48: Implementation for Socket-based system ......................................................... 196
Figure 6-49: Communication Flow on the Socket-based system ........................................... 197
Figure 7-1: Environment setup for agent and socket-based communications ....................... 202
Figure 7-2: Environmental setup for mobile devices ............................................................ 202
Figure 7-3: Measurement of TransacT (MAgSeM-based) ..................................................... 204
Figure 7-4: Measurement of TransacT (Socket-based) ........................................................ 204
Figure 7-5: Time intervals to complete a communication .................................................... 207
Figure 7-6: Generating Cipherkey and Ciphercode for MAgSeM-based system .............. 210
Figure 7-7: Generating Cipherkey and Ciphercode for Socket-based system ..................... 210
Figure 7-8: Comparison for T1 for MAgSeM-based system ................................................ 211
Figure 7-9: Comparison for T1 for Socket-based system ..................................................... 212
Figure 7-10: Comparison for T3 for MAgSeM-based system ............................................... 213
Figure 7-11: Comparison for T3 for Socket-based system ................................................... 214
Figure 7-12: Comparison for TT for MAgSeM-based system .............................................. 215
Figure 7-13: Comparison for TT for Socket-based system .................................................. 216
Figure 7-14: Execution time for TransacT for MAgSeM-based ........................................... 218
List of Tables

Table 2-1: Summary of the agent’ characteristics to handle security processes ....................... 53
Table 2-2: Summary of security technologies ............................................................................. 54
Table 2-3: List of ciphers in SSL v2 ......................................................................................... 61
Table 2-4: List of ciphers in SSL v3 and TLS v1 ........................................................................ 61
Table 3-1: Examples of Online Communication ........................................................................... 71
Table 3-2: Different Types of Information Exchanged between Users ......................................... 78
Table 3-3: Five layers of communications in MLC ...................................................................... 82
Table 3-4: Security levels excerpt from Table 7.4 from ECRYPT (2008) ................................ 92
Table 3-5: The existing key size recommendations .................................................................... 92
Table 3-6: Key size recommendation for each layer in MLC ..................................................... 93
Table 3-7: The security specifications in MLC model .............................................................. 94
Table 3-8: Ciphersuites provided by SunX509 provide ............................................................ 95
Table 7-1: Experiment setup for wired system ........................................................................ 205
Table 7-2: Experiment setup for non-mobile device agent-based system ............................... 206
Table 7-3: Time measurements for Ciphertext (MAgSeM-based) ........................................... 209
Table 7-4: Time measurements for Ciphertext (Socket-based) ................................................. 209
Table 7-5: T1 for MAgSeM-based system .............................................................................. 212
Table 7-6: T1 for Socket-based system ................................................................................... 212
Table 7-7: Percentage increased of the MAgSeM-based system for T1 ................................. 213
Table 7-8: T3 for MAgSeM-based system .............................................................................. 214
Table 7-9: T3 for Socket-based system ................................................................................... 215
Table 7-10: Percentage decreased of the MAgSeM-based system for T3 .............................. 215
Table 7-11: TT for MAgSeM-based system ........................................................................... 217
Table 7-12: TT for Socket-based system .................................................................................. 217
Table 7-13: Percentage decreased of the MAgSeM-based system for TT .............................. 217
Table 7-14: TransacT for MAgSeM-based system .................................................................. 219
Table 7-15: Transact for Socket-based system ......................................................................... 220
Table 7-16: Percentage decreased of the MAgSeM-based system for TransacT .................... 220
Table A9: DA Class..................................................................................................................268
Table A10: RA Class................................................................................................................272
Table A11: receiveMessage Class........................................................................................................275
Table A12: sSUA Class................................................................................................................280
Table A13: sDA Class................................................................................................................283
Table A14: sRA Class................................................................................................................287
Table A15: Code.java....................................................................................................................294
# List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>Agent Communication Language</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>IA</td>
<td>Interface Agent</td>
</tr>
<tr>
<td>cA</td>
<td>Crypto Agent</td>
</tr>
<tr>
<td>CBC</td>
<td>Cipher-block Chaining Mode</td>
</tr>
<tr>
<td>CDC</td>
<td>Connected Device Configuration</td>
</tr>
<tr>
<td>CLA</td>
<td>Communication Listener Agent</td>
</tr>
<tr>
<td>CLDC</td>
<td>Connected Limited Device Configuration</td>
</tr>
<tr>
<td>com_layer</td>
<td>Layer of Communication</td>
</tr>
<tr>
<td>DA</td>
<td>Decrypt Agent</td>
</tr>
<tr>
<td>DOA</td>
<td>Data Organizer Agent</td>
</tr>
<tr>
<td>FIPA</td>
<td>Foundations of Intelligent Physical Agents</td>
</tr>
<tr>
<td>FIPA-ACL</td>
<td>FIPA-Agent Communication Language</td>
</tr>
<tr>
<td>IPMS</td>
<td>Inter-Platform Mobility Service</td>
</tr>
<tr>
<td>J2ME</td>
<td>Java 2 Platform Micro Edition</td>
</tr>
<tr>
<td>JCA</td>
<td>Java Cryptography Architecture</td>
</tr>
<tr>
<td>JCE</td>
<td>the Java Cryptography Extension</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>K1</td>
<td>Symmetric key 1 (secret)</td>
</tr>
<tr>
<td>K2</td>
<td>Symmetric key 2 (shared)</td>
</tr>
<tr>
<td>KQML</td>
<td>Knowledge Query and Manipulation Language</td>
</tr>
<tr>
<td>L₀</td>
<td>Default Layer</td>
</tr>
<tr>
<td>MA</td>
<td>Mobile Agent</td>
</tr>
<tr>
<td>MAS</td>
<td>Multi Agent Systems</td>
</tr>
<tr>
<td>MAgSeM</td>
<td>Multi-agent Security Model</td>
</tr>
<tr>
<td>MLC</td>
<td>Multilayer Communication Model</td>
</tr>
<tr>
<td>MTA</td>
<td>Multi-tasking Agent</td>
</tr>
<tr>
<td>RA</td>
<td>Receiver Agent</td>
</tr>
<tr>
<td>SC</td>
<td>System Coordinator</td>
</tr>
<tr>
<td>------</td>
<td>--------------------</td>
</tr>
<tr>
<td>sDA</td>
<td>Specialized DA</td>
</tr>
<tr>
<td>sRA</td>
<td>Specialized RA</td>
</tr>
<tr>
<td>sSUA</td>
<td>Specialized SUA</td>
</tr>
<tr>
<td>SUA</td>
<td>SetUp Agent</td>
</tr>
<tr>
<td>SvA</td>
<td>Server Agent</td>
</tr>
<tr>
<td>SW</td>
<td>Social Worker</td>
</tr>
</tbody>
</table>
CHAPTER 1  INTRODUCTION

1.1 Motivations for the Research

Early technologies used for communication in the healthcare area started with bonfires to transmit information about wars and famines. After that, the telegraph was used during the Civil War and followed by the telephone and radio communications (Zundel, 1996). When the Internet network is introduced, the healthcare area has been adopting this technology and moving towards practicing healthcare services over the Internet. Electronic health or e-health uses the Internet to enhance healthcare service deliveries. Eysenbach (2001) defined e-health as:

“...an emerging field in the intersection of medical informatics, public health and business, referring to health services and information delivered or enhanced through the Internet and related technologies... the term characterizes not only a technical development, but also a state-of-mind, a way of thinking, an attitude, and a commitment for networked, global thinking, to improve healthcare locally, regionally, and worldwide by using information and communication technology.”

The Internet plays a major role for delivering services in e-health, since it offers cheap and worldwide access. Using the Internet in e-health promises to improve communication between users, because patients in rural area can access services such as consultation sessions, diagnostic aid, and remote patient monitoring (Chen et al., 2001; Eysenbach, 2001; Güler & Übeyli, 2002). In this research, the term “communication” is defined as a process of sharing

---

1 Healthcare can be defined as any care, services, or supplies related to the mental or physical health of an individual, to prevent or treat illness using services offered by the health organization. (HIPAA, 2006)
and exchanging information between two or more parties. Online communication refers to a process of sharing and exchanging information between two or more parties over a network such as the Internet. Current practices in e-health involve applications that support online communications. Sulaiman et al. (2007) compiled examples of online communications in e-health, which include videoconferencing sessions, x-ray image sharing, electronic mails, web-based applications, and also software applications used with mobile devices (such as the Personal Digital Assistants (PDAs) and smart phones) to assist mobile users.

However, although there are many Internet-based technologies developed to facilitate the communication processes and enhance healthcare service delivery, the Internet has its own drawbacks. It is exposed to security threats, which exploit the vulnerability of computer systems. The threats include network attacks, information breaches by intruders, and malicious software (malware), which is a program that is created to perform illegal operations including viruses and worms, on a computer system (Kienzle & Elder, 2003).

Pfleeger & Pfleeger (2006) defined vulnerability, threat, and attacker as the following:

- A vulnerability is defined as “a weakness in a system, for example in procedures, design, or implementation, that might be exploited to cause loss or harm”
- A threat to a computing system can be described as “a set of circumstances that has the potential to cause loss or harm”. The vulnerability of a system can be a threat to the system if it is exploited. The threat can be blocked by controlling the vulnerability. A threat contains intentions to breach a security policy. In a secure system, a security policy identifies the rules of how to control the vulnerability of the system.
- An attacker to the computer system can be defined as either hacker or cracker².

The motivation for this research stemmed from the identification of these security threats, and how these threats have caused some of the users in e-health to refuse to accept and use the

---

² A hacker is someone “who (non-maliciously) programs, manages, or uses computing systems”. A cracker is “someone who attempts to access computing systems for malicious”. However, nowadays, the term hacker is also considered as malicious (Pfleeger & Pfleeger, 2006)
technologies. Tyler (2001) reported that the security threats on sensitive medical information are among the prime factors that caused barrier to the adoption of healthcare technologies. In e-health, sensitive medical information is shared and transmitted over the Internet during a communication session. If the communications are not secured, these threats can cause severe damage to the computer systems as well as the information. The information might be modified or stolen, and thus may cause undesirable consequences. For example, if a patient’s medical information is violated or modified by a third party, the patient may not receive correct medications and furthermore, may endanger the patient’s life. These threats have raised awareness of the users, especially doctors, not to simply transmit medical data over the network without any security mechanism.

The use of the Internet in the healthcare domain has not been as widespread as the use in the Internet banking system because the users, especially the patients and doctors, were not convinced enough to use the Internet for communication purposes because of the security issues (Dearne, 2006; Dubbeld, 2006; NMRC, 2007). The doctors believed that losing the data into the wrong hands may endanger patients’ confidentiality and the consequences were hazardous (Anderson, 1996).

In the case of using the email application, doctors are aware that any medical record transmitted via an unencrypted email is like using a postcard (Dearne, 2006). A computer ‘sniffer’ program can trace the content of the email to find any important keyword. For example, the keywords name, contact number, and disease can conclude that a particular person having that particular name and contact number is suffering from a particular disease. Therefore, it is important to make sure that messages are encrypted before sending them out, especially messages that contain sensitive medical information.

There are several standards available as guidelines to safeguard the use and disclosure of protected information, such as the Health Insurance Portability and Accountability Act.

---

3 Medical information includes diagnosis, medical history, test results, current treatment, and medical prescriptions
HIPAA), the Australian Privacy Amendment (APA), and the United States Privacy Act (1974). Although these standards can be used as guidelines to handle an individual's personal details including the collection of the data and the disclosure of the information, the implementation on how to protect the data over the network is still unclear.

In summary, it is important to have a secure environment for communications in e-health in order to carry out online communications safely through the Internet. In addition, by having secure communication, users are convinced that their information is safe and their privacy is protected.

Current security technologies such as SSL/TLS, IPSec, SSH, or VPN have been robustly put into practice to provide security mechanisms to online communications. In practice, in order to use such technologies, for example SSL, one must configure the security setting and select appropriate ciphersuites\(^4\) for a particular communication. However, in this thesis, we are interested in finding a way to provide security mechanisms that can cater for different types of security needs, including different enhancements to establish more secure environments. For example, communications from a sender to multiple recipients can be done using different security strengths simultaneously, without having to reconfigure the security setting. As to our knowledge, current security technologies only provide or can only be set to one particular value of ciphersuites for every communication, that is, if one wants to have stronger or weaker security, the security must be reconfigured. We address this problem through our security model namely the Multilayer Communication (MLC), presented in Chapter 3.

Based on MLC, we aim to create a security framework, which is based on a Multi-agent system (MAS) that can cater for different security needs while providing a secure environment for online communication. We believe that MAS is a suitable approach to cater for the security processes in a distributed system. A review of MAS is presented in Chapter 4.

\(^4\)Ciphersuites is a combination of algorithms for authentication, encryption, and message authentication code (MAC), which are used to negotiate the security settings when starting a connection.
1.2 Research Questions

The aim of this thesis is to investigate the use of MAS to cater for the security processes, and to improve solutions for security issues relating to distributed communications and transactions, by proposing an agent-based security framework. The following questions provide the basis for the work in this thesis:

1. What are the standard criteria or the requirements for a secure system?
2. How can we handle the different security needs for different types of communications?
3. Can the multi-agent design framework be developed to model security needs and to design a distributed system realising the security requirements?
4. What are the criteria of the MAS approach that can be used to cater for the security processes?
5. How does MAS handle the security mechanisms for the different communications?
6. How does MAS give a much better control on security compared to the traditional non-agent based system?

1.3 Proposed Research

As stated in the motivation section, security threats that are associated with the Internet hamper a wider acceptance of Internet-based technologies in motivating the use of the e-health domain. This research is aimed at developing a security framework based on MAS to secure online communication processes in the e-health domain. The framework encompasses the MLC security model, which takes into account the different types of communications in e-health by different users, and classifies communications into a multilayered structure. Based on the layers, the appropriate security mechanisms are proposed in such a way that they will give better and flexible security strengths and performances. The MAS approach is chosen to cater for the security processes in each layer, because this paradigm is seen to be well-suited for distributed multilayered structure compared to the traditional system approach.

The system development research method will be used in this research, as the final desired outcome of this research is a prototype of an agent-based system developed to provide secure online services. We identify the roles or users in the e-health domain in order to determine the
types of message exchanges between the users. From the typical messages, we observe the
different levels of sensitivity of the information transferred during the communication, and
thus use the information as the foundation in our security model, MLC. The following list the
research components:

1. Identify security requirements for secure communication over the network/Internet
2. Identify possible security mechanisms that will be appropriate each type of
communication
3. Identify the advantages of the MAS approach and utilize suitable characteristics of
MAS to be used in developing a multi-agent system to cater for the security
mechanisms
4. Identify the way to use agents securely to transfer messages from one host to another
5. Evaluate the system to observe communication overhead and security performance, and
study the trade-off between security levels and system performances

1.4 Contributions

The goal of this research is to propose an agent-based security framework that caters for the
security processes for online communications domains like in e-health. We believe that this
research will make a significant contribution in the area of information security, which mainly
highlighted the following:

1. **The Multilayer Communication Model**

   Our approach is derived from the categorization of the types of communication commonly
found in e-health by different users. In e-health, there are many types of users that are
communicating different types of information with varying levels of sensitivity. For
example, there is information that is very sensitive that needs to be protected with high
security mechanisms. On the other hand, there is also information, which is identified as
not sensitive and does not require any protection mechanisms. The different types of
information communicated by the users, result in a classification of such communication
into a multilayered structure, which is called the Multilayer Communication Model
(MLC). Then, focus is given on securing these layers with different types of security
mechanisms. By characterising this communication into a multilayer structure, the initiator
of the communication can choose the most suitable security processes in terms of cost and efficiency.

2. Secure transmission of messages

We proposed a secure approach for a mobile agent to transfer a message over a non-secure network. The approach is designed in such a way that the user or the sender of an encrypted message can gain control over the message, by placing the relevant information needed to recover the plaintext at the sender’s side until the agent needs it for decryption purposes.

3. The Multi-Agent Security Model (MAgSeM): A security framework

The Multi-agent Security Model (MAgSeM) is a framework designed to cater for the security processes based on MLC. The MAS approach is used because of potential advantages derived from agents to cater for distributed processes (which is covered in Chapters 4 and 5). The MAgSeM-based system is able to facilitate different kinds of users’ needs in order to gain users' trust and confidence to communicate confidential or sensitive data over the network.

1.5 Thesis Organization

The thesis consists of eight chapters. Chapter 2 presents an overview of current security technologies, including the underlying network architectures and standards. Then, it examines the types of security threats to the network, as well as the existing security technologies and the mechanisms used to secure online services and communications from the threats. Next, the drawbacks of these technologies are identified and highlighted. Afterwards, this chapter introduces the field of agents and reviews the use of MAS as a solution in the security fields. From the review we identify common characteristics of the agents that are used to cater for security and communication processes.

Chapter 3 gives an in depth investigation of the background problem in the e-health domain. It studies problems that dampen the use of Internet-based technologies in e-health. It explains the need to have a secure environment for online communications. Then it thoroughly defines
and characterises the problem, provides the analysis of the problem, and presents the solution through a security model, namely MLC. It characterises communications in e-health into five categories, which represents the different types of security needs in e-health.

Chapter 4 investigates and identifies the characteristics of the agents that will be applied to support the agent systems to provide security based on our security model. It justify why a multi-agent approach is chosen in this research.

Chapter 5 is devoted to the presentation of our multi-agent based security model, or MAgSeM. It defines each agent’s goal and how and why agents are organized into layered structure. Then, it explains MAgSeM’s architecture in detail including each agent’s action, and how MLC is implemented in MAgSeM. It also presents a security mechanism based on mobile agents.

Chapter 6 focuses on the implementation of our MAgSeM-based system as well as the non-agent based systems. It includes supporting tools to develop the system in both wired and wireless environment. It also discusses the drawback of implementing the non-agent systems compared with the MAgSeM-based system.

Chapter 7 deals with the experimentation setup for this research. It evaluates and compares the finding for both systems, in terms of communication performances and overhead security. The last chapter, Chapter 8 summarises the outcomes and achievements of this research, revisits research questions and observes how far the questions have been answered. Finally, it proposes future research that may be undertaken to extend the current work.

We present two appendices at the end of the chapters. Appendix A comprises excerpts of developed programs in Java for our systems. We also present the publications made throughout the research progress in Appendix B.
CHAPTER 2 RECENT ADVANCES: SECURITY CONTEXT

2.1 Introduction

This chapter presents the background study to this research including recent technologies used in providing secure online communication. We begin with an overview of computer networks, including the underlying architecture and standards. After that, we examine the security threats and explain how the threats can jeopardize the networks. Then, examples of current security systems are presented along with their advantages and disadvantages. After that, the agent technology is discussed as motivation and support to the drawbacks of the current security systems. The advantages of the agent in the field of security are explained, and finally the research gaps are discussed in response to the inadequacies identified earlier.

2.2 Computer Networks and Security

The networked computer systems we seen today dated back to 1960s, which started with the development of minicomputers; small machines that have small processing powers to provide real-time computing results. Minicomputers also have the capability to provide a distributed processing system as it can be stationed wherever it is needed. This is in contrast with the bulky, single processor mainframe, with centralised processing mechanisms. Having a distributed processing systems means that there is a need for the computers to communicate with one another.

The operating systems were developed to manage the communication protocols that supervise the computer to computer and computer to user communications. Problems have later arisen with the compatibility issue as there were different vendors that provided different hardware and operating systems. This led to the difficulty of each computer to interoperate and internetwork with one another. In 1977, the International Organization for Standardization (ISO) had came out with a layered architecture model namely the Open System
Interconnection model (OSI) (Zimmerman, 1980). The model is concerned with how systems exchange information. It defined seven logical layers of protocols for internetworking, in such a way that it bridges the gaps between different networks with different specifications and installations requirements.

2.2.1 Open System Interconnection (OSI) Model

The seven layers in the OSI model (ISO7984-1, 1984) are Application, Presentation, Session, Transport, Network, Data link, and Physical layers.

![Figure 2-1: The seven layer of OSI model](image)

Figure 2-1 shows the seven layers in the OSI model. The summary of each layer communicating from the sender’s node to the recipient’s node is like the following (in reverse order):

i. The Application layer interacts with the user applications, and determines what kinds of resources are needed to communicate to the recipient's side. Examples of the Application layer standards are Simple Mail Transfer Protocol (SMTP), Hyper Text Transfer Protocol (HTTP), and Telnet

ii. The Presentation layer is responsible for managing the format of the data from the Application layer exchanged between sender and recipient, so that the meaning of the
data is not lost. Examples of the Presentation layer standards are ASCII, JPEG, and TIFF.

iii. The Session layer creates, maintains, and terminates communications between sender and recipient. For example, in a full-duplex operation, the Session layer allows data to go simultaneously in both sender and recipient directions, while in half-duplex operation, it only allows data to go in one direction at a time. Examples of the Session layer standards are Remote Procedure Call (RPC), NetBEUI, and Network File System (NFS).

iv. The Transport layer provides a reliable data transfer between sender and recipient. It ensures that data transfer is well established, which includes error control that allows fragments of data to be retransmitted in case of transfer failure. Examples of the Transport layer standards are Transmission Control Protocol (TCP) and Universal Datagram Protocol (UDP).

v. The Network layer is responsible for addresses and route data from the sender's side to the recipient's side. A router is commonly used to connect and interpret logical address and forward data between two or more sub network. A common network standard in the network layer is the Internet Protocol (IP).

vi. The Data-link layer is responsible for ensuring that the delivery of data over the network is error-free. The data is formed into bits, and errors from Physical layer are detected and handled. There are two sub layers, which are the Media Access Control (MAC) layer and the Logical Link Control (LLC) layer. MAC determines how a node can communicate with another node to access and forward data through a specific MAC address assigned to each computer. LLC manages the data frames flow and error checking.

vii. The Physical layer determines the physical mediums of the communication such as copper, fibre optic, or wireless.

Each layer determines rules and standards, which will be used by the next layer to complete a communication between sender and recipient.

It should be noted that not all networking standards have all seven layers like OSI. The Internet Protocol Suite, commonly known as the Transmission Control Protocol and the
Internet Protocol (TCP/IP), is a standard developed by the US Department of Defense through the work of Advanced Research Projects Agency Network (ARPANET) in the early 1970s (Clark, 1988). The TCP/IP protocol is often portrayed as layers of four protocols. The TCP/IP and the OSI models are shown in Figure 2-2.

![Figure 2-2: The TCP/IP and OSI layer](image)

The Transport layer is equivalent to layer four (Transport layer) of the OSI model, and the Internet Protocol layer is equivalent to layer three (Network layer) of the OSI model. The TCP/IP protocol is widely used and the implementation is included in most operating systems.

Although standards like TCP/IP and OSI models exist to help computers interconnect with each other, the biggest challenge is how to secure the connections or the communications, not only in the homogeneous environment such as wired connection between PC$^5$ to PC, but also in a heterogeneous environment including fat and thin client, for example from PC to a mobile phone or PDA$^6$.

---

$^5$ Personal computer or desktop computer

$^6$ Personal Data Assistant
2.2.2 Network Security Attacks

Nowadays, the emergence of computer applications that support online communications has a remarkable impact on how businesses are carried out. For example, e-services\(^7\) utilize online communications to provide services to consumers (Mehta et al., 2007; Rust & Kannan, 2002; Rust & Lemon, 2001). Applications of e-services can be seen in e-health, Internet banking, retailing, and electronic auctions.

However, these applications also bring about security issues and the needs for reliable and robust security technologies to cater for security threats, which take advantage of computer systems vulnerability.

According to Panko (2003), there are four types of security threats on the network: physical attacks, social engineering, dialog attacks, and penetration attacks. Physical attacks occur when an attacker has physical access to hardware such as the victim’s computer or network lines. For example, the attacker can intercept the transmission medium cable to get electromagnetic signals to understand the message, or tapping (Miller, 2006).

Social engineering involves the authorized staff that are vulnerable to an attacker. The attacker may deceive the staff to get information to gain access to the organization’s network. For example, an attacker could disguise as a surveyor and ask various questions to a receptionist via a telephone call. The questions might relate to the organization’s details that might be useful for hacking. The detailed methods in social engineering can be found in (Mitnick & Simon, 2002).

Dialog attacks are attacks on data in transit, such as eavesdropping to analyse traffic, impersonation, or message interception. The data in transit may include confidential information and network topology information. The attacker often uses these types of attacks to collect information on the victim’s host by using sniffing devices and programs that capture

\(^7\) Services that are delivered to users via the Internet are called e-services (Tiwana & Ramesh, 2001).
keystrokes (Marco de et al., 1998). The main purpose of dialog attacks is to gather information or to create a basis for a later attack.

Penetration attacks involve system breaches and damages to the victim’s host. The attacker will first collect the information regarding the host. According to Panko (2003), the attacker can use port scanning to understand the network, identify running services as well as specific executed programs to collect information. Afterwards, the attacker will try to guess passwords, exploit known vulnerabilities especially when the user does not control the access permission properly or does not update security and software patches regularly. Once the attacker has entered the system, he/she can perform session hijacking, that is, hijacks current communication session and impersonate as legal user with the aim of damaging the system.

Another type of penetration attack is malware (malicious software). Malware is a program that is purposely created to perform illegal operation on the computer system (Kienzle & Elder, 2003). Trojan horses, worms and viruses are examples of malware attacks that can cause heavy damage to the system. Richardson (2006) reported that in 2006, virus attacks were the prime source of financial losses in the organizations that were evaluated in their survey, and still remained the same as in 2008 (Richardson, 2008).

In e-services, users are concerned on how to provide authentication/authorization to access the information, as well as to ensure the availability and confidentiality of the information (Nahid & Shahmehri, 2001). The users, for instance in Internet banking and online auctions are reported to be wary of Internet frauds, and not confident about the security measures the sites has provided (Boyd & Mao, 2000; Costanzo, 2006; Gajanan & Basuroy, 2007). Pennanen et al. (2006) stated that users are afraid to give out their personal and financial information on the Internet because of fear that the information will be misused and exploited, and as a consequence, their privacy will be compromised. In addition, users are worried that permanent connections to the Internet create a new opening to their system, which can be exploited by hackers. The hackers might learn about their sensitive information by eavesdropping on their Internet traffics (Herzog et al., 2001).
In e-health, the main concern of using online communications is about the privacy and security of the patient’s data (Hillestad, 2008; Jennett et al., 2005; NMRC, 2007; Simon et al., 2007). For example, doctors are afraid of using unencrypted email applications to exchange messages, for fear of the information transmitted might be read or sniffed at (Caffery et al., 2007; Dearne, 2006)⁸.

The existence of the abovementioned threats are without doubt can be barriers to putting services on the Internet. A secure communication environment must first be established in order to encourage users to use the services. This includes providing mechanisms to ensure confidentiality, availability, and integrity of the information in transit, and consequently boost users’ confidence that they can communicate safely.

2.2.3 Security Architecture and the OSI Model

To satisfy the security needs for protection against system vulnerabilities and security threats, ISO has came out with another model reference namely ISO/IEC 7498-2 (1989), which defines security-related architectural concept such as security services and security mechanisms:

- A security service is defined as a measure that can be established to address a security threat.
- A security mechanism is defined as the technology that implements a security service.

The ISO/IEC 7498-2 and ITU-T X.800 (1991) recommendation classified security services and security mechanisms in OSI. The next sections list both security services and security mechanisms.

---

⁸ More on security issues in e-health is discussed in Section 3.1.2
2.2.3.1 Security Services

1. **Authentication**: a service that is used to prove the identity of a user who is communicating as valid/invalid (peer authentication), or to identify that the source of data as valid/invalid (data origin authentication)

2. **Access Control**: a service that is used with the authentication service to determine what kind of data or resources can/cannot be access to the authenticated user, in order to protect the resources from any unauthorized access.

3. **Data Confidentiality**: a service that is used to protect the data from unauthorized disclosure. This service should also able to protect data from passive attacks such as eavesdropping using sniffer program.

4. **Data Integrity**: a service that is used to verify that the data in transit is not altered or modify and that the consistency of the data is maintained.

5. **Non-repudiation**: a service that is used to verify that the sender of the data cannot deny sending it and that the recipient cannot deny receiving it. This service allows sender and recipient to provide proof of origin and proof of delivery of data.

2.2.3.2 Security Mechanisms

1. **Encipherment mechanism**: this mechanism involves encryptions of data that is the process of hiding the original data by applying mathematical functions to get scrambled data called *ciphertext*, which is the result of the encryption process. The inverse process is called decryption, which is to recover the original data from the ciphertext. This process is performed to provide confidentiality for the data. There are symmetric and asymmetric encryptions, which are based on keys used for the encryption process. Symmetric encryption uses only one key to encrypt (and decrypt) the data, while asymmetric encryption uses a pair of private and public keys to encrypt and decrypt respectively. The public key is usually publicly distributed to other users. This mechanism will be explained in detail in the Subsection 2.3.4.

2. **Digital signatures**: this mechanism is used to authenticate a particular data truly came from the right party, and that the sender cannot deny sending it. The asymmetric encryption is used in digital signatures. The sender encrypts part of the data as a token
(or signature) using a private key that only he/she can possess, and embeds the signature to the original data before sending it to the recipient. The corresponding public key is used by the recipient to verify that the signature is valid against the original data. This mechanism will also be explained in detail in the Subsection 2.3.4.

3. **Authentication**: this mechanism is used to authenticate that a user is an authorized entity, for example by using a password. Authentication can also be used through cryptography, which is using a digital certificate, where an identity is bound together with the public key that is publicly distributed.

4. **Access control**: this mechanism determines the access rights of an authenticated user including the membership of a domain name (such as in the operating system access privilege) and permission to access resources. This mechanism also provides notification of any unauthorized access for audit purposes.

5. **Data integrity checks**: this mechanism is used to verify that the data is not modified or altered. It is performed by calculating a hash value/checksum and embeds the value in the original data. The value is unique only to the original data, and normally is encrypted to avoid being stolen by any third party. The recipient can later verify the integrity of the data by recalculating a new hash value and compare it with the one sent by the sender.

6. **Traffic padding**: this mechanism adds a ‘made-up’ data to an encrypted traffic in order to conceal the real volume of data traffic. An attacker can only identify the number of packets, which flow in the traffic with the fake packet. This mechanism is used to protect the data from traffic analysis by providing confidentiality to the traffic flow.

7. **Routing control**: this mechanism is used to prevent any unauthorized party from exploiting the vulnerability of the routing protocol. This is done by predefining the route (for example through a special sub-networks or link) that the packet should be relayed.

8. **Notarization**: this mechanism uses a third party that is trusted by both sender and recipient. This mechanism is normally used with the digital certificates. The users’ certificates are bound with the third party public key, which shows that a user is trusted. This mechanism is used to assure the integrity, authenticity, and proof of delivery of the data.
We have so far identified the security threats that exist on the Internet, and how it has failed to gain a wider acceptance of technologies that are based on the Internet. We also understand that it is important to create a secure communication environment for users to gain confidence in using these technologies. Then, we have classified the security services and the corresponding security mechanisms that can be used to secure the communications against security threats.

Now we are interested in examining the existing security technologies, which are used to protect communication between two points (a sender and a recipient). In the next section, we study the types of security mechanisms used in each technology.

2.3 A Review of Existing Security Systems

In this section, we discuss current security technologies, which offer a secure environment between two communicating points. Three elements that are highlighted here are the sender, recipient, and the communication channel. For each types of the security system discussed here, we will examine the types of security mechanisms used as well as the advantages and the disadvantages of the technologies. The first two sections (Firewall and IDS) discuss security mechanisms that provide security to a network perimeter.

2.3.1 Firewall

Firewall can be defined as a device that inspects every packet that arrives at or leaves from a network to another network according to a security policy, which itself has the resistance capability to penetration (Cheswick & Bellovin, 1994). Firewalls are used to prevent any unauthorized attempt to enter the network, for example from the Internet to the internal protected network.

Traditionally, firewalls are placed between a protected network and the outside network such as the Internet; in such a way that all traffics between the two networks would pass through it (Figure 2-3).
Demilitarized zone

Demilitarized zone or DMZ is an isolated network established between the protected network and the Internet, separated by firewalls as shown in Figure 2-4. Network administrators can set up special configuration on the network, which is also called the Bastian hosts (Pfleeger & Pfleeger, 2006). Bastian hosts have very little protections and thus are set to provide limited services. They are configured as simple as possible so that attackers cannot gain network information from these hosts. The main goal is to mitigate the risk of these Bastian hosts of being compromised. However, DMZ is expected to be vulnerable and compromised in the long run and therefore, network administrators usually have a back up plan to re-establish the systems (Keromytis & Prevelakis, 2007).
Firewalls are used to protect against network attacks. This is done by performing packet inspections, application inspections, translating network address, authenticating packet, providing VPN and integrating different types of firewalls (Panko, 2003).

*Packet inspection* involves static packet filtering and stateful filtering. Static packet filtering filters packet one by one to determine which packet can be allowed to pass through by checking the header of the packet. Stateful filtering on the other hand logs the states of a connection between two nodes. Therefore, any packet segment from attackers can be traced and subsequently dropped or rejected.

*Application inspection* examines content on the packet that is not examined by the packet inspection. It uses proxy servers such as email gateways. The email gateway is located between the Internet and the protected network. All incoming emails are stored in the email gateway, which later can be retrieved by the email server (inside the protected network) periodically. The gateway does not do any active process or any other connections except waiting for connection from the email server. Therefore, if the email gateway is at risk or under attack, the attacker cannot penetrate the email server using the email gateway.

A firewall can provide an optional virtual private network (VPN) service. An external node can request for a VPN session to the firewall. Usually, the node comes from an authenticated

![Diagram of a firewall](image)
user. After negotiating a key, the subsequent sessions are encrypted and packets are relayed to the destinations node.

The disadvantages of firewalls are that, the firewalls may become bottleneck, because of the increasing speed of the network as well as the complexity of the defined policy. In addition, the complex types of the network loads increase the management problem (Keromytis & Prevelakis, 2007).

### 2.3.2 Intrusion Detection Systems

In the Information Security field, Intrusion Detection System (IDS) is commonly used to detect intrusions and alert the IDS administrator about the intrusion. It collects data on any suspicious packet for analysis. It might also pre-empt the intrusion from completing. The necessity of IDS first surfaced by the findings in Anderson’s (1980) study to improve computer security auditing and surveillance. In his study, he found three categories of attacks namely external penetration that is a penetration from an unauthorized user; internal penetration or an insider attack or attack from an authorized user who has access to the system and use it to breach the security policy; and misfeasance that is an authorized user who has legitimate access to both the system and the resources but misuses his/her right to breach the security policy.

Denning (1987) proposed a model to identify abnormal patterns and proposed a profile to represent user’s behaviours on the network, to detect normal from intrusion behaviour using statistical methods. Based on Denning’s work, the classification of IDS is formed into two categories of detections that are anomaly detection and signature detection. Anomaly detection is based on the statistical methods to identify the normal activity from intrusion. Signature-based detection involves collecting information known as profiles beforehand for normal and intrusion, and the detection is performed based on the profiles.

There are two types of IDS that is host-based and network based IDS (NIDS). As the name suggest, host-based IDS perform detection on host-basis such as the packet entering and
leaving the host, processes performed by the operating systems, and applications that are operated by the user. NIDS are placed between the external firewall that connects to the Internet or the public network and the protected network, which focus on packets that are not stopped or rejected by the firewall, as well as within the protected network. Examples of current NIDS are Therminator (Donald et al., 2002), HIDE (Manikopoulos & Papavassiliou, 2002), and SPARTA (Kruegel et al., 2001). The IDS have also been associated with anti-virus programs (Debar et al., 1999; Velissarious & Santarossa, 1999) and used to prevent unauthorized modifications to data or file structure in a system.

However, IDSs have disadvantages. As reported in Allen et al. (2000) and Lippmann et al. (2000) IDSs have mostly failed to provide reasonable detection rates, where the rate of false alarms are high. Moreover, frequent attacks on IDS have been reported, especially from DoS attacks as described in Cohen (1997) and Giovanni (2001). Cardoso (2007) added that NIDS have a high maintenance especially to update the signature and to keep the policy in place. They also argued that current NIDSs have no prevention service.

Despite all the disadvantages described above, many companies still choose to deploy IDSs as parts of their security systems. Recently, the number of companies that use IDS has grown steadily as reported by US-Computer Emergency Readiness Team (2008).

2.3.3 Cryptography

Cryptography is often used to provide protection to online communications through cryptography systems, which are software programs that implement cryptography techniques. Ferguson & Schneier (2003) defined cryptography as “the art of science of encryption”. Encryption is a process of hiding the original message (in this thesis, the original message will be called plaintext) by applying mathematical functions on the plaintext to get a ciphertext, which is the result of the encryption process. The decryption process is the reverse processes of encryption, which is to get the plaintext from the ciphertext. A pair of algorithms called cipher and a key, are needed for encryption and decryption of a message. The algorithms specify the mathematical functions or rules on how to encrypt and decrypt the message. The key consists of a piece of information such that only users who have it could encrypt and
decrypt the message. The key, algorithms, and plaintext influence the end result of the ciphertext. The relation between a plaintext, ciphertext, cipher, and a key, K can be described as:

\[ \text{Ciphertext} = \text{Encryption} (\text{Plaintext})^K \]

There are not many encryption algorithms created. The algorithms are normally publicly distributed and known to all communicating parties. Therefore, the only thing that is made secret is the key. The use of different keys produces different types of ciphertexts even though the algorithm used is the same. The key lengths determine the strengths of the encryption. The unique characteristic of key length is that, “every additional bit doubles search times” (Panko, 2003). Search times refer to the exhaustive search for the key or ‘brute-force’ attack, which involves searching \(2^{n-1}\) different keys. Therefore, the longer the key, the harder the attacker needs to try all possible keys. What attackers normally do is to capture a large amount of ciphertext to be analysed. One solution is to change the key frequently. This makes it difficult for the attackers to break the encryption, because there is not enough information of the ciphertexts produces from a certain key to be analysed within a short period of time.

There are symmetric and asymmetric encryptions based on the key used to encrypt a message. In the symmetric encryption, the same key is used to encrypt and decrypt the message. Meanwhile, two keys are used in the asymmetric encryption, which is also known as public-key encryption. One key is to be published and the other must remain secret. The use of cryptography includes hash function, symmetric key exchange, digital signature and digital certificate.

- **Hash function**
  The hash function (or checksum, or message digest) emphasizes on the integrity of the data that is to verify whether a document has been tampered. The most widely used encryption algorithms for hash function are Message Digest (MD4/MD5) invented by Rivest (1991) and Secure Hash Algorithm/Standard (SHA/SHS), designed by NIST (FIPS180-2, 2002). Hash function will take a file, calculate the hash value of the file, and that value will be stored with the file. To check whether the file has been changed, a new hash value is computed from the
file and compared with the original one. If the two values matches, then the file is considered genuine and not tampered. Two important characteristics of hash function are (1) small possibility of generating collision, which is producing the same value of hash function from different input messages; (2) the function calculation is taking into account every bit of the file being calculated, in such a way that if a single bit of the file is changed, it will randomly change the hash value. These two characteristics made hash function a useful feature in cryptography because it can produce a unique hash value while at the same time prevent an attacker to simply predict the input message. Normally, the hash function value must be secure or encrypted again so that it will not be modified by any third party.

- **Symmetric key exchange**
  The Diffie-Hellman key exchange protocol (Diffie & Hellman, 1976) is an approach of key exchange, which is said to be the pioneer of public-key exchange (Ferguson & Schneier, 2003). The Diffie-Hellman mechanism does not require the parties that want to communicate to exchange their public key in advance. To explain the protocol briefly, we consider a sender and a recipient that want to exchange messages. The sender and recipient agree in one base value called \( g \). Each sender and recipient takes one secret value that only known to them, which is \( x \) and \( y \). The sender then computes \( g^x \) and recipient computes \( g^y \). They then exchange these values with one another. Once received, the sender computes \( g^{xy} \) and recipient computes \( g^{yx} \) that literally gives the same value, which will be their shared secret value. Attacker that intercept \( g^x \) and \( g^y \) cannot retrieve the value of \( x \) and \( y \). The important part of the Diffie-Hellman protocol lies on the difficulty to calculate \( g^{xy} \), given \( g^x \) and \( g^y \). However, to use Diffie-Hellman protocol, the communicating parties need to authenticate themselves before exchanging communications to avoid attackers impersonating their true identities.

Key exchange can also be performed with the public key encryption, which is often used along with the symmetric encryptions. In the public key encryption, a pair of key (public and private key) is used to allow two parties that want to communicate to share a symmetric key, in such a way that it cannot be intercepted by attackers. Consider a sender and a recipient with their key pairs \((pubKs, privKs)\) and \((pubKr, privKr)\), and assume that they have exchanged their public key in advance. The public key cryptography can be used like the following procedures:
(1) The sender encrypts a plaintext with key $K$, and produce a ciphertext. Then $K$ is encrypted with his/her private key $E(K)_{privKs}$. Later the recipient can decrypt the key using the sender’s public key: $D(K)_{pubKs}$.

(2) The sender encrypts a plaintext with $K$. Then, $K$ is encrypted with the recipient’s public key $E(K)_{pubKs}$. The recipient later decrypts it with his/her private key $D(K)_{privKs}$. Therefore only the recipient can retrieve the key.

In (1) however, any user that has the sender’s public key can decrypt the message as well, which is not a good way to exchange the key. In (2), the recipient does not know that $K$ is from the sender.

One way to solve this is by encrypting the key with the sender’s private key, and then encrypting it again with the recipient’s public key. As a result, only the recipient can decrypt the message and at the same time the recipient can make sure that the message came from the sender by verifying it with the sender’s public key.

- **Digital Signature**

Digital signature is used to authenticate that the data came from the right party and the sender cannot deny sending it, or non-repudiation. The public key encryption can be used to produce signature. Normally, the digital signature is created by hashing a plaintext $P$, and the result of the hashing process is encrypted with a private key of the sender. Consider $M$ is the hashed message derived from a plaintext, $P$. The sender signs $M$ by encrypting $M$ with his/her private key $E(M)_{privKs}$. The recipient can later decrypt it back using the sender’s public key $D((E(M)_{privKs}))_{pubKs}$. Therefore, the recipient knows that the message must come from the sender (authenticity).

The recipient can save the $E(M)_{privKs}$ for future references, in case that the sender try to say that he/she did not send $P$. The recipient can take $P$ and $E(M)_{privKs}$. Using $pubKs$, retrieve $M$ from $E(M)_{privKs}$. Then recalculate a new $M$ from $P$, and compare the new $M$ with the original one. If both are the same, the sender cannot deny sending it because $M$ has come from the sender and only the sender’s private key can produce $E(M)_{privKs}$ (non-repudiation).
However, a question arises on how the recipient knows that he/she is using the correct public key of the sender, out of many other users who have exchanged their public keys with the recipients. This is where the term Digital Certificate comes in.

- **Digital Certificates**

Certificates are often used in the public key infrastructure (PKI), where the public key that is publicly distributed is bound together with an identity or information of the user as a digital signature, and attached to a digital certificate. Users have two options, either by self-signing the certificate or by using a trusted third party (TTP). By self-signing the certificate, a recipient can check the true identity of the sender, although it offers little guarantee (in an organization, the certificate can be issued by the system administrator). If the TTP is chosen, the sender must make sure that the TTP is also trusted by the recipient. TTP validates the information of the sender and recipients’ identities before issuing certificates. By using the digital certificate, users can make sure of the authenticity and integrity of the message, as well as ensuring non-repudiation.

Next, we discuss cryptography systems that use cryptography techniques to provide protection to online communications.

### 2.3.4 Cryptography Systems

A cryptographic system is a software program that provides security functionalities based on cryptography technique to the communicating parties. Normally, the communicating parties do not realize the underlying security processes of the system. In this section, we discuss existing cryptographic systems that are used to secure our day-to-day online communications.

#### 2.3.4.1 The Secure Socket Layer (SSL/TLS)

The Secure Socket Layer (SSL) was originally developed by Netscape Corporation ([http://netscape.aol.com/](http://netscape.aol.com/)) to provide protection on its browser. Later, SSL was standardized and known as Transport Layer Security (TLS) (Dierks & Allen, 1999). SSL/TLS works on the transport layer of the OSI model, described in Section 2.2.1; which means, it protects traffic in
the application layer. In general, SSL’s goal is to provide a secure channel between the sender and recipient

- **Initial handshake:** both sender and recipient negotiate on a *cipher suite* that is a set of cryptography algorithms that will be used in the communication session. The cipher suite is a composition of the public key mechanism such as RSA, a symmetric cipher (block cipher such as RC4, Triple DES, AES, IDEA, or DES) and hash algorithm such as MD5 or SHA and their associated key size. The fact that there were weak algorithms such as RC4 and DES is because they are listed as the default values. This issue should be noted when configuring the cipher suite, as an attacker might exploit the initial handshake process into using weak encryption to transfer message (Hook, 2005).

- **Authentication:** For communication between sender and recipient, both parties are required to exchange certificates so that they can authenticate each other.

- **Key exchange:** key exchange is performed after the authentication process succeeds. A symmetric key is selected at the sender’s side. This key will be the session key for the communication. To exchange the key, the sender uses the recipient’s public key in the certificate to encrypt the session key, and send it to the recipient’s side. The recipient then decrypts the session key using its private key. Afterward, the two parties will use the key to encrypt the message in the subsequent communications.

Although SSL/TLS does not provide security automatically to an application that wishes to benefit from the SSL/TLS functionalities⁹, the deployment of SSL/TLS continues to grow at a robust rate. For an attacker to hack and break into the SSL/TLS channel is still a costly effort. There has not been any reported case of sensitive credit card number in transit being hacked while being protected by the SSL/TLS channel (Panko, 2003).

---

⁹ To deploy SSL/TLS, the application must be specifically programmed to be SSL/TLS aware.
2.3.5.2 IP layer security (IPSec)

IPSec (RFCs 2401-2411 and RFC 2451) provides security protection to the Internet layer\(^{10}\), and this protects all IP data packets regardless of the protection given on the application layer and transport layer. No modification or reprogramming of applications is needed if IPSec is used. IPSec uses two protocols to provide security protections, which are the Authentication Header (AH) and Encapsulating Security Payload (ESP).

AH provides data integrity and authentication of origin of the IP packets. The authentication process is based on MAC, using HMAC algorithm (Krawczyk et al., 1977) and a secret key. The value of MAC is calculated from the message to be sent and embedded in the header as specified in RFC 2402 (Kent & Atkinson, 1998a). This value will be recalculated once the packet has arrived at the destination host. If both values are the same then the packet is considered authenticated. The reason AH is used although without confidentiality of the data (where the attacker that sniffs the data can read it) is because there are countries that does not support encryption for data confidentiality (Panko, 2003) when sending data across another country.

ESP on the other hand provides full confidentiality through an encryption process and an optional authentication. ESP provides an encryption mechanism to encrypt IP packets before being transmitted to the receiver host and there the packets are decrypted. This provides confidentiality to the data and prevents any eavesdropping to the data. Various types of algorithms are supported by IPSec for encryption performed by ESP such as Triple DES (Data Encryption Standard), RC5, IDEA, CAST, and Blowfish. Although the RFC 2406 (Kent & Atkinson, 1998b) specifies the use of DES for encrypting the packet, however, DES is known to be a weak algorithm and it will be replaced to AES in the future (Anirban & Manimaran, 2007). The authentication process in ESP is slightly different than the one in AH. The authentication in ESP involves authenticating the IP packet payload, while in AH, it only authenticates the IP header.

---

\(^{10}\) Refer to Figure 2-2
IPSec has two modes of operation, which are the Transport Mode and the Tunnel Mode. The Transport Mode provides host-to-host security over public networks such as the Internet. Figure 2-5 shows the Transport Mode operation. The IPSec header is placed after the original IP Header.

![Figure 2-5: IPSec Transport Mode](image)

The IP Header in the Transport mode is not encrypted because it is needed to route the packet to the recipient’s host. The drawback of this mode is that an attacker can learn the destination of the packet from the IP header that is not encrypted.

Meanwhile, Tunnel Mode does not provide host-to-host protection. Instead, it provides security only between IPSec gateways that connects two different networks such as shown in Figure 2-6. Because the messages are transmitted between gateways, the original IP header can be encapsulated within a new IP header. This is done by adding IPSec header and a new IP header to the original IP packet. As a result, the original destination of the packet, which is the recipient, can be hidden away from the attacker that analyse the traffic. Once arrived at the recipient gateway, the gateway dismantles the message to recover the original IP header, and then routes the message to the recipient’s host according to the IP number. This mode does not require the user to install any additional software to the computer. However, the IP packets routed by the gateway in the internal organization network are not protected by any security mechanisms. Therefore, any disgruntled employee can simply launch any attacks on the packet.
Figure 2-6: IPSec Tunnel Mode

- **Security Association (SA):** Security Association (SA) describes the relationship between sender and recipient(s) that specifies the supported types of algorithms and security protocols that can be used to secure the communication sessions. In a one-way relationship, if the sender wants to send a message to the recipient, one SA is needed, while in a two-way relationship, which has two-way communication between the sender and recipient, two SAs are needed, and so on.

- **Key Management:** Key management in IPSec is managed by the Internet Security Association Key Management Protocol (ISAKMP) defined in RFC 2408 (Maughan et al., 1998). For every security session established, a shared key must be used. In IPSec, the ISAKMP is executed through the Internet Key Exchange (IKE), which is used to provide a standard for authentication, negotiation of security services, and generating a shared key. Diffie-Hellman is used for the key-exchange protocol that exchanges a shared secret. With this shared secret value, the communicating parties authenticate each other and exchange the secret key and choose their SA.

Stallings (2000) reported that strong security can be provided to all traffic if IPSec is used in a firewall or a router. In addition, no traffic overhead is produced within an organization. He added that because IPSec is transparent to users, the organization does not have to educate the
employees on how to use IPSec. However, due to the complexity shown in IPSec, IPSec needs extra precaution and careful deployment.

2.3.5.4 Virtual Private Network (VPN)

VPN is a private and secure connection established from two connected networks from sender to recipient over the Internet. Business organizations can also benefit from VPNs, because remote employees are able to connect to the corporate network anywhere using the Internet, which is cheaper compared to the expensive long-distance dial-up ISDN and leased lines (Schneider et al., 1993). In addition, VPNs provide scalability, where new connections can be easily added regardless of geographical distance; and flexibility, where new services can be integrated into VPN. Furthermore, VPNs can be configured to have many levels of security in the same network by changing the encryption strength (Northcutt et al., 2005).

Generally, in the IP-based VPN, it works by tunnelling IP packets by adding a new header to the packet, so that it can be encrypted and authenticated. Then, at the receiving end, the packets are assembled to the original form. The receiving end can be firewalls, routers, gateway, or hosts. Pardoe & Snyder (2005) classify VPN into three categories that are the intranet VPNs, remote access VPNs, and extranet VPNs.

Intranet VPNs involve establishing a private network in an organization intranet and does not allow any access to the public network or the Internet. This type of VPN is used to secure critical applications such as financial systems, sales or customer database, and information exchange between departments in an organization. Because of the high-speed network of the intranet, fast and strong encryption is required in this type of VPN. Remote access VPNs use the Internet network to connect remote departments or employees with the main office within an organization. This type of VPN requires reliable network and services, as well as high level of scalability to manage the various VPN connections to the main office. Extranet VPNs use open systems\textsuperscript{11} to support interoperability to connect to the external entity of the organization

\textsuperscript{11} systems that involve different types of network including the Internet, different hardware and software
such as business partners, customers, or suppliers. Strong authentication is used in extranet
VPNs to prevent unauthorized access to the VPN sessions.

- **Tunnelling Protocols:** The Internet Engineering Task Force (IETF) has proposed a
  number of tunnelling protocols for VPN. The types of tunnelling selections are based
  on their placements in the OSI layers.

1. **Point-to-Point Tunnelling Protocol (PPTP)**
PPTP takes place at the Data-link layer and uses TCP port 1723. It encapsulates PPP
packets and transmits the packets through a tunnel over a public IP network. PPP uses
authentication protocols such as Password Authentication Protocol (PAP) and
Microsoft Challenge-Handshake Authentication Protocol (MS-CHAP). PPP provides
confidentiality on the data by providing encryption using DES and 3DES. PPTP itself
does not provide any security, as it does not encrypt the encapsulated packets and just
provide the tunnel medium (Hiemstra, 2003). However, MS-CHAP has vulnerability
because of its weak password policy (Schneier et al., 1999).

2. **Layer Two Tunnelling Protocol (L2TP)**
L2TP extends the Cisco’s proprietary Layer 2 Forwarding (L2F) tunnelling protocol
described in RFC 2341 (Valencia & Littlewood, 1998) combined with the PPP to add
interoperability to the L2F tunnelling protocols. L2TP does not encrypt data nor
authenticate messages. L2TP also occurs in the Data-link layer and uses UDP port
1701. Townsley et al. (1999) provide details on L2TP.

3. **IPSec and SSL/TLS**
IPSec can be incorporated into VPN, and is treated as the best proposed tunnelling
protocol for VPN because of its end-to-end security protection (Doraswamy &
Harkins, 1999). SSL/TLS can also be used to provide a private network not only to
browsers and web servers, but also can be integrated with other protocols. For instance,
protocols like SMTP that is used to transmit emails can be integrated with SSL from
client applications. Eudora is an example of client email application that provides optional SSL service.

The disadvantage of VPN is that it cannot protect any unauthorized attempt to enter a network because it only protects the data stream created for a particular private communication. Therefore, VPN cannot protect the whole network. Since examining the packet flows through IDS or traceroute tools is ineffective against VPN, securing the network is difficult because the organization has no control over packets that enter the network through VPN. The VPN tunnels might provide backdoors to the network if their employees do not have a secure PC at home (Northcutt et al., 2005).

2.4 Security Applications

There are various aspects that have been catered for in the security field, such as from monitoring the security at the network perimeter, for instance firewalls and IDSs; securing the hosts inside the network, such as personal firewalls and anti-viruses; to securing communications between hosts, such as SSL, SSH, IPSec, and VPN as has been discussed in the earlier sections.

We centred our discussion in this section to the security mechanisms applied to _application systems_ that support online communications or message exchanges via the Internet. Communications or message exchanges via the Internet have commonly been seen in various fields from peer-to-peer communication, social networking to banking and businesses, such as in e-services.

We now take a look at security mechanism in e-services. E-services can be characterized into five categories (Mehta et al., 2007; Tiwana & Ramesh, 2001), the first three are for gaining profits:

1. B2C (Business to Consumer): business organizations that provide services directly to its customers such as Internet banking and retailing via e-commerce;
2. C2C (Consumer to Consumer): consumers that provide services to other consumers such as in electronic auctions that involve a trusted third party to act as a mediator;
3. B2B (Business to Business): a business organization provides services to other business organizations, such as supply chain management.

Another two is for non-profit purposes:

4. G2C (Government to Consumer): any government agencies that provide services to the consumer, such as e-health and e-tax;
5. G2B (Government to Business): any government agencies that provide services to business organizations, such as online voting.

The operations of e-services without doubt require message exchanges, which involve critical information to be included in the transaction. For example, in B2C and C2C, users of Amazon.com and eBay.com will need to enter credit card details in order to purchase any products (Mehta et al., 2007; Tiwana & Ramesh, 2001).

Mehta et al. (2007) and Tiwana & Ramesh (2001) gave an overview of the current security mechanisms to provide protection in e-services, listed like the following:

1. User ID and password are commonly used for authentication.
2. SSL is used for data confidentiality, while MD5/SHA1 is used for data integrity.
3. Email applications are used for an immediate confirmation for non-repudiation
4. Privacy policy is provided for privacy protection.
5. Certificate authorities (CA) that are trusted by both users are used to provide a chain of trust in the e-services environment. CA provides certificates, which bind together the public key of the owner.
6. Anonymity using an independent party is used to provide authentication to the user, or no authentication is given at all.

Other mechanisms include logging every misbehaving user through the IP address to provide user location traceability, and using a third party services such as credit card verifier before deciding to provide the service. However, Mehta et al (2007) argued that it is rather difficult to implement these security mechanisms in mobile devices because of their low processing power.
The commonly used security protection for mobile device that uses wireless LAN are user authentication and encrypted wireless network such as WPA\(^{12}\) (Ahmad, 2003; Bowman, 2003; Johari, 2009; Wi-Fi, 2005). In addition SSL is also used on wireless devices to provide transport level security (Gupta & Gupta, 2001; Marti et al., 2004).

There are also proprietary software tools that support online communications such as demonstrated by GoToMyPC (http://gotomypc.com), Groove (http://www.groove.net), and Waste (http://waste.sourceforge.net/).

*GoToMyPC* uses multiple layers of strong passwords as authentication, data confidentiality using SSL with AES-128 bit (FIPS197, 2001), and end-to-end authentication. *Groove* uses password-based authentication; DES (Mehuron, 1999) and AES-192 bit algorithms to provide data confidentiality on disk; as well as to data over the network to provide end-to-end security. Data integrity is also provided using hash message and message authentication code. *Waste* uses TLS to provide data confidentiality and builds a web-based PKI for trust between the users.

In the medical domain, security is realized by integrating it with the applications of information systems of the health organization. This is to provide authentication, confidentiality, and availability to the information, using cryptography protocols to encrypt, decrypt, sign, and hash messages (Bleumer, 1994; Marković, 2006; Merger et al., 1997). SSL is also used to established a secure network for information exchanges (Marković, 2006; Polemi & Marsh, 1998; Smith & Eloff, 1999; Ulieru & Ionescu, 2004).

However, the existing security approach presents a number of drawbacks. It is common that end users do not necessarily have any deep knowledge of security. They usually are not aware of any security provided (if any) to them. For example, medical staff are not entirely security-aware, and if they are performing transactions on the Internet without proper knowledge, they

\(^{12}\) WiFi Protected Access
may end up allowing malware or phishing\(^{13}\) activities to occur. Moreover, in the case of mobile device users, to successfully secure the devices, proper training is needed to familiarize users with the security mechanisms.

Another disadvantage is that, the level of the security provided is not flexible and cannot be changed according to the organization’s need. This is because the security configuration in the security mechanisms, such as SSL is set to provide a fixed security to the user per communication session. If there is a change in the organization’s need for a higher security level for example, the SSL needs to be reconfigured at the security setting (this will be detailed in the next Section 2.6.1.2).

To overcome these weaknesses, we are interested in studying the Multi Agent System (MAS) approach and how the agent-based systems could provide better techniques to enhance the traditional non-agent based systems. For a start, agent can be specialized with certain skills to handle tasks on behalf of users (Wooldridge, 2002; Yee, 2006). This is especially useful to help users who are lack of security skills to cater for the security processes. The agent can be used to assist users to lessen the complexity of the tasks. Agents are used to provide user interfaces and perform tasks on behalf of the users (Bergenti et al., 2002; Enembreck, 2003).

To provide a rather flexible security protection to users, agents can be used to cater for the security processes. An agent with specialized skills can be instantiated and added to the system without configuring the whole system. This is particularly useful to solve security problem with different security features. For example, in an agent-based security system, a sender can choose to have a particular security mechanism for a communication with say, Recipient A. The sender can also be allowed to choose different security mechanism for a communication with Recipient B. To accomplish this feature, an agent equipped with AES 128-bit key can be instantiated to provide 128-bit encryption strength. The same class of agent can be instantiated and equipped with AES 256-bit key to provide 256-bit encryption strength.

\(^{13}\) Phishing is a process done by fake web sites that purposely created to mimic the legitimate business or financial institution to steal information (by making users to reveal their personal information)
As a result, the sender can be provided with a flexible security protection according to users’ preferences, without having to reconfigure the whole system. Thus, users can opt for lower security strength to be used in any low processing power device.

Agents can automate processes without or with minimal user intervention, by having the autonomy characteristic. It allows agents to take the initiative to do a task, without having to wait for the user or any other entity to invoke them. For instance, a mobile agent can carry secure messages across platforms, and then perform the security processes at the platforms without having the user to interfere with its processes.

Besides having the autonomous ability, agents can also portray cooperative and communicative abilities. These abilities allow the agents to coordinate with each other to achieve a given goal (Novák et al., 2003; Pitsillides et al., 2006; Sycara, 1998; Ugurlu & Erdogan, 2005). This is particularly useful for a distributed system with distributed processes.

The characteristics of the agents will be discussed in details in Section 4.2. In the next section, we discuss in further details about agents and the characteristics they display in order to support the traditional non-agent based systems.

2.5 Agents and Security

We have discussed briefly in the previous section, how MAS characteristic can be used to perform processes on behalf of users, cooperate with each other to handle and automate the security processes, and can be instantiated with different skills and knowledge where necessary. Now we continue our discussion, with our approach of interest, the agent technology and the advantages of this technology to the security field. This section describes the agent technology and the characteristics of the agents. Then, it discusses the use of these characteristics to support the traditional non-agent based approach to cater for the security processes.
2.5.1 Agents

There is no general accepted definition for agents and multi-agent (Bradshaw, 1997; Jennings et al., 1998). Researchers normally define agents in the context of their research. Bradshaw (1997) defines agent as “a software entity which functions continuously and autonomously in particular environment”. Jennings et al. (1998) define agent as a computer system that can receive input and output from the environment, autonomous and flexible. Wooldridge & Jennings (1995b) describes agents by their characteristics, which are weak notion and strong notion. Weak notion includes autonomy (able to perform tasks without direct intervention from user), social ability (able to interact with each other), reactivity (able to perceive and give respond to the environment) and pro-activity (able to exhibit goal-directed behaviour by taking the initiative). Strong notion includes weak notion of agent with AI characteristics, including mentalistic notions like knowledge, beliefs, intention, and obligation. It also aims to model agents with emotions that have human-like mental states. These definitions given by the authors suggest that agents live together in an environment and collaborate with each other in order to collectively achieve certain goals.

2.5.2 Agent Characteristics

![Figure 2-7: Agency properties](image-url)
In spite of the many definitions of agents, we agree with the definition of Wooldridge and Jennings’ (1995b): “an agent is a computer system that is situated in some environment, and that is capable of flexible and autonomous actions in this environment in order to meet its design objectives”, in which, flexible means that the system must have responsive, proactive, and social characteristics (Jennings & Wooldridge, 1998).

Figure 2-7, which is depicted from Garcia et al. (2002), described the agent’s properties. According to Garcia et al. (2002), the fundamental properties of the agents include autonomous, interactive, and adaptive; other properties are treated as complementary for the agency. The agent’s properties are described as the following:

- **Autonomous:** Agents are autonomous, which has goals and states. It has control over its own actions without any intervention from users and cannot be invoked like a method. The state is driven by beliefs, goals, capabilities and plans. An agent controls its life cycle by having its own thread of execution and decides when to perform the actions (Garcia et al., 2002; Jennings & Wooldridge, 1998; Wooldridge & Jennings, 1995b).

- **Interactive:** Agents communicate by sending messages using Agent Communication Language (ACL). There are two most well known ACLs that are Knowledge Query Manipulation Language (KQML) (Finin et al., 1997) and FIPA-Agent Communication Language (FIPA-ACL). The agents are either communicating with the environment or with the other agents.

- **Adaptive:** the ability of an agent to adapt or adjust its state when receiving messages from other agent or its environment (OMG, 2000).

- **Learning:** the ability of the agent to learn from historical information and respond to its environment

- **Coordinative:** Agents have their own roles when coordinating with each other to allocate resources and improve efficiency using their expertise in a shared environment. Agents encapsulate its task and functionality within its body, and interact with other agents based on a well devise plans (Garcia et al., 2002; Huhns & Singh, 1998; Jennings, 1993).

- **Cooperative:** Agents are able to coordinate with other agents to achieve a common goal. This is also defined as collaborative (Weihmayer & Velthuijsen, 1995).
• **Sociable:** Agents have the ability to perform two-way communication using a communication language to cooperate by exchanging knowledge, belief, and plans to solve given tasks, all of which are beyond an individual capabilities (Wooldridge & Jennings, 1995a, 1995b).

• **Intelligent:** Agents are able to express knowledge such as belief, goals, and reasoning and to interact with other agents using symbolic language (Oliveira et al., 1999; Wooldridge & Jennings, 1995b). Bradshaw (1997) defined intelligent as the “degree of reasoning and learned behaviour”, that is, how the agents learn and adapt the environment.

• **Mobility:** Agents are capable of migrating from one environment to another to carry data along with its code to be executed remotely (Appleby & Steward, 1994; Bradshaw, 1997).

The characteristics of the agents described above, when combined together can lead to a solution designs amenable for distributed problem solving by creating a system involving multiple collaborative agents.

### 2.5.3 Multi-agent Systems and characteristics

Agents can be categorized into two, which are single agent systems and multi-agent systems (Weiss, 1999). A single agent system, such as an information searching agent, is launched to search for information based on the owner’s request preferences. However, single agent system is unsuitable for a distributed problem domain, because it does not assist in reducing the complexity of the problem, and it does not use any of the characteristic described earlier. Agents that are unable to work with other agents are likely to become virtually useless (Flores-Mendez, 1999). Agents need to interact with each other in order to achieve specified goals. Oliveira et al. (1999) defined MAS as “a collection of, possibly heterogeneous, computational entities, having their own problem solving capabilities, and which are able to interact among them in order to reach an overall goal”. Each agent has its own capability or specialty to do tasks. This is what makes the agents suitable to represent and cater for problems that have multiple problem solving methods and entities (Jennings et al., 1998).
According to (Jennings et al., 1998; Sycara, 1998), the characteristics of MAS can be listed as the following:

1. each agent has incomplete capabilities or specialization to solve problems, that is, MAS is suitable to handle complex problems by using modularity or problems that can be divided into sub-problems;
2. no global system control, that is, each agent has its own control over its given tasks;
3. data is decentralized, such that, each agent will have to negotiate and coordinate among each other in order to obtain resources; and
4. computation is asynchronous, the processes of each agent are independent of each other.

The use of MAS to solve problems is motivated by the following criteria (Sycara, 1998) (Stone & Veloso, 2000):

1. The problems are too large to be solved by a single massive architecture. MAS is seen as one possible solution because of its capability to provide modularity. Interactions between agents can solve problems with conflicting goals. MAITS (Sharma et al., 2007b) can be an example of a large security problem that is managed by MAS. MAITS caters for system monitoring, authentication/authorization, IDS, and forensic analysis. Each task is managed by a group of agents organized on a predefined hierarchy based on functionality.
2. MAS can solve problems that require multiple independent components interacting with each other to complete given tasks. This is precisely what MAS can do, that is to skill the agents with different capabilities or specialization and interact with each other to achieve certain goals (Hannon & Burnell, 2005).
3. MAS permits interconnection/interoperation of multiple legacy systems, that is, to integrate the legacy systems with MAS, so that the systems can utilize and fully exploit the agent’s capabilities. Genesereth and Ketchpel (1994) suggested an agent wrapper to integrate the legacy systems with MAS.
4. MAS can solve problems by using a distributed information source such as in (Clark & Lazarou, 1997).
5. MAS can solve problems where the expertise is distributed such as in the education domain (Triantis & Pintelas, 2004).

6. Performance of the systems can be improved with:
   a. parallel or concurrent processes are seen as advantages as it could enhance system operations. Each process is becoming less complicated, wherein the programmer can easily assign tasks to different agents;
   b. reliability, where MAS has redundant agents that can tolerate failure by sharing controls and responsibilities among different agents;
   c. extensibility, an agent that represents new a service can easily be added to the system without being reconfigured, or reprogrammed as seen in the legacy systems;
   d. robustness, MAS works well in unpredictable situations;
   e. maintainability, MAS is easy to maintain because of the modularity, and it is easy to solve small sub problems than to take the problem as a whole;
   f. flexibility, agents with different specialization and knowledge are able to organize and perform tasks with other agents in an environment
   g. responsiveness, an error that occurs within a subtask will be handled locally, and thus will not be spread to the whole systems;
   h. reuse, an agent with specialization and knowledge can be ‘unplugged’ from one system and ‘plugged in’ to another system so that the system can exploit the agent’s ability

2.5.4 MAS and Security

The concept of agents has become widely accepted and studied to address the diversity and complexity of many real world problems. The applications of agents can be divided into two parts, which are applications in distributed systems and applications in personal software assistants (Wooldridge, 2002).
2.5.4.1 Example of MAS applications

The applications of agents include various types of domains. Examples include:

- information retrieval such as WebCrawler search engine
- air traffic control, where agents are used to represent aircrafts and air traffic control system such as in OASIS (Ljungberg & Lucas, 1992);
- electronic commerce that uses agents to manage and serve users requests such as in buying and selling services such as Kasbah (Chavez & Maes, 1996);
- business process management, which is to skill agents to manage services, business task and resources, as well as perform business operations such as found in (Reichert & Dadam (1998) and Wenger & Probst (1998);
- medical domains such as to assist and supervise medical protocol executions (Alsinet et al., 2000), supports communications and collaborations of different users for homecare DITIS (Pitsillides et al., 2006), and assist diabetes patients at home (Haan et al., 2005); personal agent, to assist users to lessen the complexity of the user tasks (Bergenti et al., 2002; Enembreck, 2003); and
- security field, for example in IDS (Balasubramaniyan et al., 1998; Chan & Wei, 2002; Lui et al., 2005; Zhang et al., 2001).

In this research however, we are narrowing down our discussion to examine MAS for security purposes. We investigate how MAS and the characteristics of agents can be used to improve solutions to security issues relating to distributed communications and transactions.

2.5.4.2 MAS security approach

Recently, MASs have emerged as one of the most promising solution to cater for the complex, robust, and reliable security processes. (Ferber, 1999; Stone & Veloso, 2000; Sycara, 1998; Weiss, 1999) have suggested that MASs are well suited for applications that are distributed, complex, modular, scalable and flexible, where problems in the security fields exhibit these characteristics.
There have been a number of studies that discussed the characteristics of MASs, which make them appealing to be employed to cater for the security problems. (Alsinet et al., 2000; Ametller et al., 2004a, 2004b; Poggi et al., 2001; Sharma et al., 2007b; Ugurlu & Erdogan, 2005) have presented various advantages and benefits of MASs, which are autonomous, mobility, scalability, extendibility, modularity, flexibility, and mitigating complexity problems. Multi-agent systems provide a platform to develop an architecture that can benefit security problems by enhancing the security’s reliability, flexibility, robustness, and maintainability.

These studies involve using MASs to handle the security services and security mechanisms such as discussed in Section 2.2.3. To have a secure system environment, security services and security mechanisms must be defined and implemented. Security services like authentication, access control, data confidentiality, data integrity, and non-repudiation (ITU-X.800, 1991) are often described as security requirements for secure systems (Bleumer, 1994; Jansen & Karygiannis, 1999; Pfleeger & Pfleeger, 2006).

1. **MAS to provide secure services**
MAS can be used to provide secure services to user. Shakshuki et. al. (2004) presented a generic multi-agent architecture that is capable to provide secure services to users. The agents cooperated together to support security tasks such as authentication and authorization. The agents also communicated with the user to get input, display output, and provide security processes such as encryption, decryption, verification and digital signature. Each agent is skilled with certain knowledge to complete a certain task, such as:
   1. an agent interacts with the user to take input/show output, it also take user’s input and send it to other agent;
   2. an agent to authenticate a user, that take the user’s ID and password. The neural network approach is used for the agent to reason and learn to validate a user
   3. an agent to authorize the user, that is when the user is proved to be authorized, the agent performs a decision making process to decide what resources that the user can access; and
4. agents that provide security services such as encryption/decryption, digital signature, and verification.

The agents are autonomous where it has its own goal and state. In addition, interactivity and cooperative characteristics are the main features presented in their research, where the agent exchange and share information and results among each other. Apart from the cooperative and autonomous features, there is an intelligence characteristic integrated with the agent to reason and make decision.

![Diagram](image)

**Figure 2-8: Certification and information delivery phase**

MAS can also be used to provide security to application-specific services, such as proposed by Alsinet et. al. (2000). They modelled medical services in hospitals as specialized domain agents (SDA), which are named as Surgeon Agent or Nurse Agent. Figure 2-8 and Figure 2-9 are excerpts taken from (Alsinet et al., 2000), to describe the security processes.

The idea is to secure the message transfer between SDAs when executing medical protocols (MP). Certification Agent (CA) provides certificate to other SDAs. Certificates are for distributing each SDA’s public key. Supervisor Agent (SA) traces every interaction between SDAs. Medical protocol server agent (MPSA) provides information such as a list of available MPs to SDA and SA. The message to be transferred is secure based on a socket approach such
as in SSL protocol. After the certificates are exchanged, each SDA opens a socket, and execute a handshake using parameters of the ID and private key of the agent, as well as the public key of the CA.

A symmetric session key is created to encrypt the messages. All messages are encrypted and signed. Received messages are authenticated using digital signature, and thus, these mechanisms provide message authentication, confidentiality, and integrity of the message.

All SDAs must communicate with CAs to obtain certificates of other agents. The agents show a cooperative characteristic where SDAs and DAs cooperate with each other in order to execute MPs. SA is needed to monitor and assist the SDAs activity. For every SDA communication, SA is created to provide a list of possible and forbidden actions that are to be performed by SDAs. SA will record any anomalous decision performed by SDA. Each agent is autonomous, that is it has its own goal and state, as well as interactive.

Another application-specific example is the matchmaking application named Yenta (Foner, 1996, 1997), that used MAS approach to provide secure services to users with similar
interests. Users that have Yenta systems can connect to each other. For each user, a Yenta agent is permitted to scan emails and files owned by the user, and find and categorize the user’s interests into a group of granules. The user is able to remove any granule that is not preferable. The Yenta agent will evaluate the granules and try to join a group that has other Yenta agents in. In the group, the agent sends message to the group, introduce itself, and find other that have similar interest (Foner, 1997).

Each Yenta agent provides security in terms of data confidentiality by using an IDEA/DES symmetric key, PKI for managing keys and providing public key cryptography, and hash functions to provide integrity to the message. Yenta agent is autonomous, it has its own goal and state and can perform tasks without the intervention of the user. In addition, Yenta agent is interactive and cooperative as it will communicate and exchange information with other Yenta agents to find whether the other users represented by the other agents are suitable or a perfect match for its owner.

2. Securing inter-agent communications

Generally, to provide secure services to users, the communication between agents must first be handled. Many researchers developed architectures to secure inter-agent communications, which is either intra- or inter-platform communications. Novák et. al. (2003) presented an architecture and implementation of X-security system including authentication and secure inter-agent communications. The system issues certificates to each agent to authenticate the agent and executes security processes. There is a Security Certificate Authority (SCA), shown in Figure 2-10, which is an excerpt from (Novák et al., 2003), and each agent must register its certificate to SCA. A certificate stores the corresponding agent's information and is signed by SCA. An agent can request a certificate of other agents if the certificate is not in its list. If SCA is not responding to the request, the agent is allowed to directly ask the intended agent. Each agent has a security module that lists all algorithms (symmetric and asymmetric) and registers it to SCA.
A sender agent needs to ask the receiver agent about the supported algorithms, and then the security module of the sender agent generates a session key (based on the supported algorithm) and sends it to the receiver agent encrypted using public key cryptography. The subsequent message is then encrypted with the session key and digitally signed. As soon as the communication is terminated, the session key is deleted or invalidated. The X security system agents are autonomous, interactive, and sociable.

Wong & Sycara (1999) also proposed a secure communication between agents, which is based on the RETSINA system (Sycara et al., 1996), which has an Interface Agent that interacts with users, Task Agent that performs tasks specified by the user, and Information Agent that manages the information sources, shown in Figure 2-11, which is taken from (Sycara et al., 1996).

RETSINA also has infrastructure entities, which are the Agent Name Server (ANS), which manages agents’ IDs and addresses; and the Matchmaker, which caters for agents ID and their capabilities. Other agents communicate with ANS and Matchmaker for agent’s addresses and capabilities.
To secure agent communications and provide trust, they add security elements to both ANS and Matchmaker. Both of them have their certificates trusted and publicly distributed by other agents. Each agent is given a certificate. An agent can verify another agent’s certificate through the Agent Certification Authority (ACA). The ACA’s role is to verify a certificate using an agent’s ID and the public key, verify certificate requests from other agents, and revoke the agent’s public key.

For communications between an agent with ANS, or an agent with Matchmaker, certificates and Public key cryptography are needed in order to authenticate the message. Each agent signs a message with its private key, so that the message can be verified using the agent’s public key. For message confidentiality, all communications are secured using SSL. RETSINA agents are autonomous, interactive, cooperative and sociable. Authentication, confidentiality, and integrity are provided to agent communications.
3. Mobile Agent Security

![Figure 2-12: Self-protected mobile agent mechanism](image)

A security mechanism is proposed in (Ametller et al., 2004a, 2004b; Pedro Manuel et al., 2006) to provide mobile agent security, where the agent itself carries the protection mechanisms without depending on the sender/owner's platform. Figure 2-12, which is taken from Pedro Manuel et al. (2006), illustrates this mechanism. Their goal is to protect the fragment of code owned by the mobile agent \(d\), which must only be executed at the intended recipient’s platform. In order for the recipient to verify that \(d\) is not modified, the hash of \(d\), \(H(d)\) is generated and signed to produce \(S\). Then \(d\) and \(S\) will be encrypted with a symmetric key \(r\).

There is a control code \((C)\), which will be the mobile agent’s main code used to communicate with the recipient’s agent. However, to execute \(C\), the recipient must first authenticate that \(C\) is indeed came from the sender. Therefore, the sender generates the hash of \(C\), \(H(C)\) and encrypts it with the recipient’s public key \(K_j\), together with \(r\). Therefore, at the recipient’s platform, only recipient can retrieve \(H(C)\) and \(r\) with his/her private key. \(C\) can be verified using \(H(C)\), and if it is valid, it will be executed to perform a request to a service from the platform to decrypt the message.

Afterwards, \(r\) is used to decrypt the code \(d\) and \(S\). The signature \(S\) is generated using a disposable key pair \((Pa, Sa)\). The sender’s agent signs the data using the private key \((Sa)\), which yield \(S\). The corresponding public key \((Pa)\) is sent to the recipient’s host together with
S, which is encrypted with the recipient’s public key. The recipient’s agent can check the authenticity of d by verifying S with Pa. The agent is autonomous because it has its own goal and state, interactive as well as cooperative when dealing with other agents at the other hosts.

Ugurlu & Erdogan (2005) exploited Java language (Arnold & Gosling, 1998) architecture to provide ‘isolation’ to mobile agents in SECMAP, described in Figure 2-13, an excerpt taken from (Ugurlu & Erdogan, 2005). A Secure Mobile Agent Server (SMAS) is launched at each computer host to create and dispatch agent. SMAS created a new agent by instantiating a private object of its AgentShield class. As the agent is declared as private it gave the effect of encapsulating and preventing the agent to be directly accessed by other agents.

Communications of the agent are performed through an interface object: AgentInterface. An agent can request to migrate or communicate with other agents through the interface. Each SMAS has its own certificate, to encrypt/decrypt data. Thus, the data came from SMAS can be authenticated and verified. Each agent’s code and state are always encrypted using DES, and can only be decrypted when it is in the running state.

Figure 2-13: SECMAP architecture
The migration agent’s code is also encrypted, until the agent arrived at the destination host using the DES key retrieved from the Security Manager. SMAS has three instances or mode called Standard mode (S-SMAS) that carries out standard operations such as create, activate, inactivate, remove, and permission to communicate or migrate; Master Browser mode (MB-SMAS) that manages agent IDs and locations; and Security Manager mode (SM-SMAS), which maintains security related components. The communications between agents are secured by SSL established using the certificates own by the SMASs, and thus, authentication, confidentiality, and integrity of the data can be provided. The agents are autonomous and interactive. Communications occur between the agents and SMAS. The mobile agent migrates to another platform to perform tasks on behalf of the user, and therefore projecting a mobile characteristic.

Table 2-1 summarizes MAS applications in security. From the table, the most used characteristics of agents to cater for the security processes are autonomous, interactive, and cooperative to support the inter-agent communication and cooperation in order to achieve the agent’s goals. Mobility and other agent’s characteristics can be integrated to the multi-agent systems according to needs. We are interested to investigate these characteristics further in order to apply them in our proposed approach, which will be further detailed in Chapter 4.
<table>
<thead>
<tr>
<th>No</th>
<th>Author</th>
<th>Agent’s characteristic</th>
<th>Security service</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(Shakshuki et al., 2004)</td>
<td>Interactive, cooperative, autonomous, and intelligence</td>
<td>Authentication, authorization, data confidentiality</td>
</tr>
<tr>
<td>2</td>
<td>(Alsinet et al., 2000)</td>
<td>Interactive, cooperative, and autonomous</td>
<td>authentication, confidentiality, and integrity</td>
</tr>
<tr>
<td>3</td>
<td>(Foner, 1996)</td>
<td>Interactive, autonomous, and cooperative</td>
<td>authentication, confidentiality, and integrity</td>
</tr>
<tr>
<td>4</td>
<td>(Novák et al., 2003)</td>
<td>Interactive, autonomous, and sociable</td>
<td>authentication, confidentiality, and integrity</td>
</tr>
<tr>
<td>5</td>
<td>(Wong &amp; Sycara, 1999)</td>
<td>Interactive, autonomous, sociable, and cooperative</td>
<td>Authentication, confidentiality, and integrity</td>
</tr>
<tr>
<td>6</td>
<td>(Ametller et al., 2004a, 2004b; Pedro Manuel et al., 2006)</td>
<td>Interactive, autonomous, mobile, and cooperative</td>
<td>Authentication, confidentiality, and integrity</td>
</tr>
<tr>
<td>7</td>
<td>(Ugurlu &amp; Erdogan, 2005)</td>
<td>Autonomous, interactive, cooperative, and mobile</td>
<td>authentication, confidentiality, and integrity</td>
</tr>
</tbody>
</table>
2.6 Gaps in Security: Motivations

We will now revisit the current security technologies that provide protection to online communications as discussed in Sections 2.3.4 and examine what are the limitations and the inadequacy portrayed by these technologies.

2.6.1 Different Communication Needs: Current Technologies and Limitations

Table 2-2 presents the summary of the technologies. We are interested in the technologies that can secure online communications between two points, in other words, the security mechanisms used to secure data transmission between two points. Referring to Table 2-2 above, SSL/TLS, SSH, IPSec, and VPN are suited for this purpose and have provided robust security mechanisms to secure online communications.

<table>
<thead>
<tr>
<th>Security technology</th>
<th>Mechanisms provided</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL/TLS</td>
<td>Protect the transport layer Provide authentication, digital signature, and encryption to the message in transit using a collection of algorithms in the negotiated cipher suite.</td>
</tr>
<tr>
<td>SSH</td>
<td>Protect the transport layer Provide authentication, data integrity, and digital signature. Data encryption is provided during SSH session, that exchange a symmetric key.</td>
</tr>
<tr>
<td>IPSec</td>
<td>Protect the Internet layer Provide authentication, data integrity, authentication of origin, and anti replay protection. Data confidentiality is provided using symmetric key algorithms.</td>
</tr>
<tr>
<td>VPN</td>
<td>VPNS provides various tunnelling protocols such as authentication, data encryption, data integrity, and digital signature</td>
</tr>
</tbody>
</table>

In general, each technology offers a list of available and supported symmetric algorithms that are used to encrypt messages in transit during the communication sessions. However, as we have discussed in Section 2.4, these technologies do not cater for different types of
communication needs in an organization. For example, consider that an organization uses SSL for its secure communications. If the organization needs to change the security strength of the SSL channel\textsuperscript{14} to be stronger or weaker, it cannot be flexibly provided to the organization. The person in charge, for instance the Security Administrator, needs to reconfigure the systems to change the security setting.

The need for stronger or weaker security strengths is necessary in information classification standard, such as portrayed in ISO 17799, where distinguished level of security protection is needed for different types of information with different levels of sensitivity. In an organization, different types of communications carry different types of messages. These messages contain different types of information with different levels of sensitivity.

We are motivated to find the best way to secure these different types of communications in such a way that it could provide different types of security strengths to the communication, which can flexibly be selected by the user. The next section discusses the different types of information as well as information classification in further detail.

2.6.1.1 Different types of information with different levels of sensitivity
In this section, we introduce the concepts of sensitive information and the level of sensitivity of the information in greater details. Sensitive information are those that should not be revealed to public (Pfleeger & Pfleeger, 2006). Whether the information is considered sensitive, is based on the importance or the values of the information, and who is communicating it. It is important for an organization to decide whether the information will cause a significant loss to the holder if it is made public.

For instance, a communication that exchanges information such as a name, a place, and a meeting time, are less sensitive than information that has a name, an address, and types of diseases that are considered more sensitive. A third party that intercepts this conversation may correlate the information and conclude that a person with that name and address has that

\textsuperscript{14} The security strengths indicate the key lengths of the ciphers. Refer to Table 2-3 and Table 2-4
particular type of diseases. Such information, if revealed to public will cause embarrassment and loss of reputation to the patient.

“The desired degree of secrecy about such information is known as its sensitivity” (Economic-Expert, 2009) such as more sensitive or less sensitive. The level of sensitivity of the information can also refers to the degrees of loss or potential damage to the holder, if the information is disclosed to a party that does not have any authority to access it.

The levels of sensitivity of information often relate to the classification of sensitive information. Classification of sensitive information can be seen adopted in most governments and business-related organization around the world. Classification of information is considered important because it provides guidelines to (1) classify certain information to different levels of sensitivity, and (2) protect information from any unauthorized access by providing a distinguish level of security protection to the information.

There are existing standards for information classification. ISO provides information classification guideline in ISO 17799, which classify information as Top Secret, Highly Confidential, Proprietary, Internal Use Only, and Public Documents. Each of these classifications categorizes different types of information with different levels of sensitivity. The verbatim definition of each criterion is as follows (ISO17799):

i. Top Secret: Highly sensitive internal documents and data. Has very restricted distribution indeed, and must be protected at all times. Security at this level is the highest possible.

ii. Highly Confidential: Information which is considered critical to the organization’s ongoing operations and could seriously impede or disrupt them if made shared internally or made public. Security should be very high.

iii. Proprietary: Procedures, project plans, operational work routines, designs and specifications that define the way in which the organization operates. Used by authorized personnel only. Security at this level is high.

iv. Internal Use Only: Information not approved for general circulation outside the organization, where its disclosure would inconvenience the organization or
management, but is unlikely to result in financial loss or serious damage to credibility/reputation. Security at this level is controlled but normal.

v. Public Documents: Information in the public domain. Security at this level is minimal.

US government categorises, sensitive information as Top Secret, Secret, and Confidential. Australia and New Zealand governments have an additional criterion known as Restricted. The verbatim definitions of the information classification are as follows (EO12958, 1995; SIGS, 2001):

i. Top Secret: the unauthorized disclosure of which reasonably could be expected to cause exceptionally grave damage to the national security that the original classification authority is able to identify or describe.

ii. Secret: the unauthorized disclosure of which reasonably could be expected to cause serious damage to the national security that the original classification authority is able to identify or describe.

iii. Confidential: the unauthorized disclosure of which reasonably could be expected to cause damage to the national security that the original classification authority is able to identify or describe.

iv. Restricted: Compromise of information would be likely to affect the national interests in an adverse manner.

In summary, from the classifications, we can imply the following:

a. ‘Top Secret’ is the most sensitive information, or

b. ‘Highly Confidential’ information is more sensitive than ‘Proprietary’ information, or

c. ‘Confidential’ information is less sensitive than Secret information.

We can also imply that more sensitive data has greater degree of loss or potential damage compared to the less sensitive data.

2.6.1.2 Communication Scenarios

Suppose that a company needs different security levels of protection to secure different kinds of communications. Consider that the company has distributed employees, as well as business partners, and customers. Different users that are involved with the company have different
goals and purposes when communicating with other users. Examples of these users are described in Figure 2-14, which are the top management members of the company, financial department, financial department of a remote branch, ICT department, business partner A, as well as Customer A.

Communications in each department or across departments can occur either within the company’s internal network or involving external networks such as the Internet. There is a need within the company, that for each communication, different security mechanisms are required so that only the most sensitive information can be secured with the highest level of security mechanisms, medium sensitive information can be secured using medium level of security mechanisms, and information that is considered as low sensitive, but still cannot be revealed to others can be secured using allow level of security mechanisms.
For example, a top management might want to send two different messages to two different users:

1. A message containing a secret about business strategic plan to another user in the top management level;
2. A technical problem in his/her personal computer to a user in the ICT department.

If considering the company’s need to secure the message according to the sensitivity of the message, these two different messages require two different sensitivity levels. The first message can be considered as most sensitive, and the other is low sensitive. These two messages must be secured differently, the first with the highest level of security mechanisms and the second with low level of security mechanisms.

In Section 2.3.3, we have described that the key lengths in the symmetric key encryption determines the strength of the encryption, and thus represents the security level or security strength that can be provided to secure the communication. However, with the current technologies, these different types of security levels cannot be applied to the different types of communications in the example of communication scenarios described above. This is because current technologies only allow all communications sessions to be secured with the same security strength. This symmetric key is selected during the configuration or set up phase. If one wants to change the security levels of the communication, one needs to reconfigure the setting.

1. Security strengths and configuration in SSH

Figure 2-15 shows an example of symmetric algorithms or ciphers that are used to give security strengths in the SSH sessions (SSH, 2003).

\[\text{security strength} \]  

\[\text{security level} \]  

\[\text{security strength} \]

15 The most sensitive, more sensitive, and low sensitive used here is as discussed in Section 2.6.1.1

16 Security level and security strength will be used interchangeably throughout this thesis
The ciphers are selected from the first drop-down menu, or alternatively they can be manually inserted by typing the list of the supported ciphers. The following is an excerpt taken from (SSH, 2003) for the instruction of choosing the ciphers:

- **AnyCipher**: Any available cipher (apart from none) can be used.
- **AnyStdCipher**: Allows only standard ciphers, i.e. those ciphers mentioned in the IETF-SecSH-draft (excluding none). This is the default cipher value.
- **AES128**: Use 128-bit Advanced Encryption Standard (Rijndael) encryption.
- **AES192**: Use 192-bit Advanced Encryption Standard (Rijndael) encryption.
- **AES256**: Use 256-bit Advanced Encryption Standard (Rijndael) encryption.
- **3DES**: Use 3DES encryption.
- **Blowfish**: Use Blowfish encryption.
- **Twofish**: Use Twofish encryption.
- **Arcfour**: Use Arcfour encryption.
- **CAST**: Use CAST encryption.
- **DES**: Use DES encryption. DES is generally considered a very weak cipher, and its use is not recommended. It is offered as a fallback option only.
- **none**: Don't use encryption. Use this option for testing purposes only!
2. Security strengths and configuration in SSL

The same feature is portrayed in the SSL configuration. Table 2-3 and Table 2-4 from IBM (2009) represent the list of ciphers supported by SSL version 2, SSL version 3 and TLS version 1. The way the cipher is selected to encrypt the messages in transit is by choosing the first cipher in the server's predefined list that are supported by the client's machine (IBM, 2009).

Table 2-3: List of ciphers in SSL v2

<table>
<thead>
<tr>
<th>Short name</th>
<th>Long name</th>
<th>Meaning</th>
<th>Strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>27 SSL_DES_192_EDE3_CBC_WITH_MD5</td>
<td>Triple-DES (168 bit) RC4 (128 bit)</td>
<td>stronger</td>
<td></td>
</tr>
<tr>
<td>21 SSL_RC4_128_WITH_MD5</td>
<td>RC2 (128 bit)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23 SSL_RC2_CBC_128_CBC_WITH_MD5</td>
<td>DES (56 bit)</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>26 SSL_DES_64_CBC_WITH_MD5</td>
<td>RC4 (40 bit)</td>
<td>weaker</td>
<td></td>
</tr>
<tr>
<td>22 SSL_RC4_128_EXPORT40_WITH_MD5</td>
<td>RC2 (40 bit)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24 SSL_RC2_CBC_128_CBC_EXPORT40_WITH_MD5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2-4: List of ciphers in SSL v3 and TLS v1

<table>
<thead>
<tr>
<th>Short name</th>
<th>Long name</th>
<th>Meaning</th>
<th>Strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>3A SSL_RSA_WITH_3DES_EDE_CBC_SHA</td>
<td>Triple-DES SHA (168 bit)</td>
<td>stronger</td>
<td></td>
</tr>
<tr>
<td>35b TLS_RSA_WITH_AES_256_CBC_SHA</td>
<td>AES SHA (256 bit)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35 SSL_RSA_WITH_RC4_128_SHA</td>
<td>RC4 SHA (128 bit)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34 SSL_RSA_WITH_RC4_128_MD5</td>
<td>RC4 MD5 (128 bit)</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>2F TLS_RSA_WITH_AES_128_CBC_SHA</td>
<td>AES SHA (128 bit)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39 SSL_RSA_WITH_DES_CBC_SHA</td>
<td>DES SHA (56 bit)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62 TLS_RSA_EXPORT1024_WITH_RC4_56_SHA</td>
<td>RC4 SHA (56 Bit)</td>
<td>weaker</td>
<td></td>
</tr>
<tr>
<td>64 TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA</td>
<td>DES SHA (56 Bit)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33 SSL_RSA_EXPORT_WITH_RC4_40_MD5</td>
<td>RC4 MD5 (40 bit)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36 SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5</td>
<td>RC2 MD5 (40 bit)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32 SSL_RSA_WITH_NULL_SHA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31 SSL_RSA_WITH_NULL_MD5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30 SSL_NULL_WITH_NULL_NULL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. Security strengths and configuration in IPSec
For IPSec, the same case also occurs in the configuration process. When selecting the cipher, Figure 2-16 describes the default Security Methods for IPSec setting on Windows XP Professional, which includes the ciphers and the hash algorithms. The user can select the default setting, or add a new security method as needed.

![Figure 2-16: IPSec setting on Window XP Professional](image)

For SSH, SSL, and IPSec cipher selection described above, only one cipher is chosen for a communication session. If using the example of communication scenarios in Section 2.6.1.2, where a top management user wants to send messages to two different people with different level of security mechanisms, this user needs to use another cipher a with different security level by reconfiguring the *cipher* or *ciphersuite* field. In other words, current technologies do not cater for the following requirements:

1. Provide different security strengths to secure different types of communications.
2. Provide an automated and a flexible way for the system to cater for all kinds of users’ needs without reconfiguring the system, in other words.
3. Provide mechanisms to handle security for two communication users in different environments, such as PC to PDA communications.

2.6.2 Motivations for secure multilayered communication structure

In order to satisfy the requirements that current technologies are lacking, we focus on developing a security framework, which can cater for different types of communications with different levels of sensitivity, by providing different types of security mechanisms suited for the communications.

As discussed in the Section 2.6.1.2, the communication scenarios given suggested that there can be more than one type of communication in the company. We could classify these communications into groups, based on the different levels of security provided to secure different levels of sensitivity of the information.

The idea is comparable to the one in (IBM, 2009), to classify SSL cipher (such as described in Table 2-3 and Table 2-4) and based on three types of the key lengths classifications, which are HIGH, with key lengths larger than 128-bit; MEDIUM, with key lengths equal to 128-bit; and LOW, with key length smaller than 128-bit. However, this classification is limited to only three classifications, which cannot accommodate information classification such as modelled in (ISO17799) or in (SIGS, 2001).

Yee (2006) proposed an approach to personalize a security policy that is suited to a user before communicating using e-services. The personalization includes the selection of combinations of the following security mechanisms: Authentication, Access Control, Data Confidentiality, Data Integrity, Non-repudiation, Secure Logging, Certification, Malware Detection, and Application Monitoring. Users can choose their own security policy according to preferences, which allow them to choose their own security mechanisms to provide low, medium, or high security protection to be chosen in their security policy. There are simple, negotiable, automated and a combination of negotiable and automated security policy options.
The simple security policy contains a low security mechanism; negotiable security policy allows users to negotiate their own security policy with the service provider, whether they want a low or high security mechanism; automated security policy uses agents to negotiate the policy according to the user’s preferences; and the combination security policy allows users to negotiate the policy and then be automated by an agent. The simple security policy is useful for users with less security-related knowledge. However, for the other options, this approach needs users with fairly knowledgeable skills in security-related fields.

In a research to secure online transmission of medical data, performed by Doupi et al (2005), they classified the data being transmitted based on either transmission at local-level, regional-level, or cross-border level. The security measurements are higher at the cross-border level, compared to the local-level and regional-level. The security technologies used depended on the latest technologies available. However, the classification of users’ communication was not considered in this research. Thus, all data which was transmitted at every level has the same security measure for all users.

Suppose that there are more than three types of information with different levels of sensitivity. Thus we need more than three types of communications, and as a result, we need more than three types of security mechanisms. In this situation, we believed that characterizing the communications into a layered structure is the best way to cater for the security processes. A layered structure can be seen in information classification such as described in (ISO17799) and (EO12958, 1995). ISO provides the information classification guideline in ISO 17799, which classifies information as Top Secret, Highly Confidential, Proprietary, Internal Use Only, and Public Documents. For governments like the US government, sensitive information is categorized as Top Secret, Secret, and Confidential. Each of the elements represents the sensitivity levels of the information it carries. In addition, the level of security that must be applied to the information is also stated. Figure 2-17 portrays the ISO 17799 in a layered architecture. The top layer represents the most sensitive information, while the lowest layer represents the lowest sensitive information.
We could adopt this characterization concept of information classification to classify online communication. Then, we can organize and apply security mechanisms with security levels appropriate to each layer. With the use of a multilayered structure it can lead to several advantages, for example modularity: security mechanisms can be captured independently based on the policy defined at every layer; and flexibility: any element of security mechanisms can be added or removed systematically when necessary, for example, we can add or remove a cipher with a certain key length to/from the layer. Chapter 3 will detail the layered structure that will be used in our approach. However, before going deeper into the layered structure, we are interested in employing MAS to cater for the security processes in the layered structure.
2.6.3 Multi-agents for secure multilayered communication

There are many criteria that should be highlighted in order develop a secure environment for communications that support different types of communication needs. For a communication to be secure, security processes must be provided either at the user level or application level. For example, checking user’s credentials for authentication processes; applying cryptography protocols to the information; asking a recipient’s permission to send information across, and securing the communication channel to send the information.

For security systems that have distributed processes (for example, waiting for the recipient’s answer to send a message), the processes need to collaborate together to share and exchange information to achieve the specified goal. In addition, in the layered structure (discussed in Section 2.6.2), each layer has its own distributed processes that have to be catered for in order to provide protection to the layer. Moreover, if we are taking into account the heterogeneous environments the users are in, for instance a user working on his/her PC communicates with a user working on his/her PDA, we will have to consider the implementation of the security processes for different types of operating systems and devices. Therefore, we need a mechanism that can cater for these distributed security processes.

The multi-agent system approach is used in this research because of the abilities of the agents to work together in an environment to handle the security processes. The ability of the agents to autonomously perform tasks and at the same time being cooperative among each other is seen as the best solution for the multilayered structure. In addition, other characteristics such as mobility, extensibility, or intelligence can be added if needed.

2.7 Summary

This Chapter has presented the overview of the network, the security attacks and how the security threats can compromise the communication networks. Recent security technologies as well as its advantages and limitations have also been presented. Then, we present agent technology and the advantages of this technology as a problem solver.

17 Such as discussed in Section 2.5
The limitation of current technologies to provide different kinds of security needs to the users, combined with the multi-agent system characteristics to especially cater for the security processes have provide motivations for the development of a secure multilayered communication model. The next chapter discusses in detail, our layered structure model, namely the Multilayer Communication model (MLC).
CHAPTER 3 PROPOSED MULTILAYER COMMUNICATION MODEL TO SECURE E-HEALTH COMMUNICATIONS

3.1 Introduction

This chapter starts with giving an overview about current online communication practices in e-health to deliver healthcare services. Then, it discusses the security issues that hinder the acceptance of e-health technologies supporting the communication. After that, using e-health communication as the motivating problem, we identify the problem characteristics and analyse the problem. We then propose a model, namely the Multilayer Communication Model (MLC), to cater for the security processes in online communication. This model classifies online communications in e-health into five layers according to the levels of sensitivity of the information being transmitted during the communications. Our MLC model is motivated by recent observation of the problem of providing flexible security mechanisms to online communication discussed in Section 2.6.

3.2 Users and Networks

3.2.1 Online Communication

The prevalent use of the Internet these days has influenced the use of online communications in many areas such as the healthcare, education, military, financial, and business. In the healthcare area, online communications are currently being exercised to be part of medical practices. The next section discusses the types of communication tools or technologies that use the Internet to support online communications.
A video conferencing session is used as part of the communication tools in the hospital. For example, the Renal Unit staff at the Queen Elizabeth Hospital in Adelaide communicate with other staff from another hospital to exchange information and discuss issues regarding renal transplantations using images captured from previous operations (Mitchell, 1999). Apart from video conferencing, the email technology is also used as a tool for online communication. Shou et al. (2005) reported that the use of emails eased the consultation processes among patients and doctors. Since the use of emails in e-health has become widespread, guidelines of email communication in e-health are provided, such as presented in AMA (2000) and Sands (1999). The guidelines are used to define appropriate use of the email as a medium of communication and describe how to use the email technology to its fullest potential.

Online communication can also be performed through web-based applications to exchange messages. Wen and Tan (2003) reported that web-based applications provide a structural way to organize information. Users can systematically access their information and reduce working time. Liederman et al. (2004) reported that the web-based messaging is a better way to communicate compared to standard email applications, because the messages can be automatically categorized and routed to appropriate staff. For instance, requests for appointment, billing, medication, and online consultation can be redirected to the scheduling department, accounting department, nurses, and doctors respectively. As a result, medical staff, like doctors, may no longer be overwhelmed with inappropriate patient email messages, which require more time to manage. Pagliari et al. (2005) developed a web-based e-health application that focused on communication between medical staff and patients. The application supported online services such as the delivery of electronic test results, requests for appointment, electronic transfers for hospital discharge or clinical letters, electronic referrals, and clinical emails. Tang et al. (2003) developed a quite similar architecture of an e-health application like Pagliari et al., which has functionalities like patient-doctor online messaging, access to health summaries or test results, prescription renewal requests, appointment requests, and billing requests. The system they developed was integrated with the patient’s electronic medical records and provided a convenient way for patients to access their information.
Through these functionalities, communication with web applications can save time and provide an efficient way to deal with any non-urgent communication with patients.

Wireless devices are also used to support users in e-health. For example, the nurses in the Royal District Nursing Service in Victoria, Australia are provided with PDAs and smart phones to help them to get information through emails while moving from one house to another (Mitchell, 1999). Other wireless devices include wearable devices worn by patients such as the portable electrocardiogram (ECG), which can be monitored remotely by the GSM cellular network (Scanlon, 2000).

From this discussion, we can observe that there are different types of tools used to support online communication in e-health. Table 3-1 summarises these online communications.

<table>
<thead>
<tr>
<th>User</th>
<th>Tools</th>
<th>Two-way Communications</th>
<th>Type of information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Renal Unit Staff</td>
<td>Video conferencing</td>
<td>between renal unit staffs from different hospital</td>
<td>Renal transplantation</td>
</tr>
<tr>
<td>Doctors, patients</td>
<td>Email</td>
<td>Doctors and patients</td>
<td>Consultation</td>
</tr>
<tr>
<td>Scheduling and accounting departments, nurses, doctors</td>
<td>Web-based application</td>
<td>- Scheduling department and patients</td>
<td>Billing, medication, consultation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Accounting department and patients</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Nurses and Doctors</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Nurses and Patients</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Doctors and Patients</td>
<td></td>
</tr>
<tr>
<td>Nurses, patients</td>
<td>Wireless devices</td>
<td>- Nurses transfer message from server to mobile devices</td>
<td>Patient’s information</td>
</tr>
</tbody>
</table>

We can also observe different types of users involved in the communications such as doctor, patient, nurse, and general staff (such as for billing and accounting departments) depending on the organization’s needs. There are also different types of information involved and exchanged between users using different technologies, such as surgical procedures, patient-doctor consultation, medical information, patient’s diagnosis and test results, requests for
appointment, and billing. Some information can be considered as more sensitive or less sensitive\textsuperscript{18} than the other. For example, the patient’s medical information, diagnosis, and test results are more sensitive than appointment requests and billing requests. Regardless of the sensitiveness of the information, the technologies used in e-health are without doubt helping to enhance the effectiveness of the communications in e-health.

However, while these technologies greatly facilitate and enhance the service deliveries, the security threats to these technologies also come in parallel, which can compromise the patient’s privacy. As a consequence, not all users in e-health for example doctors and patients can accept the use of these technologies to deliver healthcare services. In order to share the information, the system and the network must be secured to gain user's confidence and trust. In the next section, we discuss the motivation for secure communication in further details.

\textbf{3.2.2 Motivation for a Secure Communication Environment}

Nowadays, computer technologies are mushrooming very fast and have changed how people do tasks. People often use these technologies to communicate and share information. The Internet is used as a back bone to connect people and communicate online. The reliance on the Internet as a medium of communication and for sharing information is so prevailing. The Internet is not only used in the medical domain, but also in other fields such as in the education, military, business corporations, and financial institutions. As a result, information has become the prime priority in our daily lives. Some information will be useful; other information however can be harmful if it fell into the wrong hands. For example, if sensitive medical information of a particular patient such as critical diseases or medical history is stolen by an intruder, the information can be used to threaten the patient.

The Internet allows borderless access to information regardless of geographical location, physical ability, economic capacity or social status. However, security issues relating to

\textsuperscript{18} Comparison between more sensitive and less sensitive of the information can be seen in Chapter 2, Section 2.6.1.1.
threats\(^{19}\) from the Internet hamper a wider acceptance of Internet-based technologies. These attacks not only can be done by a professional attacker, but also by any individual that has the intention to learn how the attack is done and later perform it, because the anonymity provided by the Internet allows people to exchange the methods of hacking and cracking, and also provides instructions for doing it (Spatscheck & Peterson, 1999).

In e-health, the widespread use of technologies to provide healthcare services was hindered by the concern about privacy or security of the data (Hillestad, 2008; Jennett et al., 2005; NMRC, 2007; Simon et al., 2007). For instance, doctors are not interested in using email applications to exchange message if it is not secured, as fear of the information transmitted might be read or ‘sniffed at’ (Caffery et al., 2007; Dearne, 2006). The distributed nature of disseminating critical information, especially through the unsecure network such as the Internet, without proper security measure will compromise the privacy and put the owner of the information at risk from an uninvited attacker. In addition, Lofstrom & Moerschel (2008) revealed that current practices of sharing and exchanging the information through the Internet with external organizations were performed without proper security practices.

There is also a need to protect mobile health or mHealth users, which involve users with mobile devices (Gururajan, 2004). Although there is an increasing adoption of mobile devices reported in 2010 (Versel, 2010 ), little security measures has been taken into account, which caused a rise in concern among medical practitioners (Miliard, 2010; Versel, 2009). Information exchanges via mobile devices “*needs to be protected against incidental or deliberate disclosure to third parties*” (Skulimowski, 2004).

The security issues listed above are the motivating problem in e-health communication. It is therefore important to carry out any communication process from a secure environment. The term “*secure*” is referred to as security goals (Panko, 2003; Pfleeger & Pfleeger, 2006), which address the aspects of *confidentiality, integrity, and availability*.

---

\(^{19}\) As discussed in Chapter 2, Section 2.2.2, the prime security threats on the Internet network are classified as physical attacks, social engineering, dialog attacks, and penetration attacks (Panko, 2003)
- **Confidentiality**: to prevent any unauthorized user from accessing the information.
- **Integrity**: to prevent or detect modification of the information.
- **Availability**: to ensure that the data can be accessed by authorized users. This implies that the right of the authorized users to retrieve the data cannot be denied.

The Confidentiality, integrity, and availability aspects are the important essences that should be taken care of, when identifying the security requirements for a secure system.

### 3.3 The Problem

In general, the motivating problem in e-health that we have discussed so far can be viewed as a class of security problem in a distributed architecture. This section onwards discusses the problem in further details. We start with investigating the problem characteristics and analysing the solution for a secure communications. We use the e-health problem as the main example, however, the solution presented here will not only suitable for e-health domain, but also provide a proof-of-concept solution to other domains that require a secure online communication environment.

#### 3.3.1 Problem Characteristics

The characteristics of the security problem centred on the current implementation of secure online communication among geographically distributed users. The implementation includes how communications are secured in the homogeneous and heterogeneous distributed environment, such as communication from a wired network (e.g. from PC to PC) and from a wireless to a wired network (e.g. from a PDA to a PC). In our approach of securing online communication, we are taking into account the different levels of sensitivity\(^\text{20}\) of the information that is transmitted by the users. In the next sections we will investigate further the types of information and the sensitivity levels of the information that is transmitted during communications in e-health. Then, we proposed an information classification based on ISO17799 to classify the information in e-health. This information classification will be the base of or proposed MLC model, which will be discussed in Section 3.2.2.

\(^{20}\)For levels of sensitivity, refer to Chapter 2 (Section 2.6.1.1)
3.3.1.1 Identifying the Sensitivity Levels of the Information

In Section 3.1.1, we have identified different \textit{types of users} in the online communication and we have also identified \textit{different types of information} with different levels of sensitivity. There is information that can be considered as \textit{more} sensitive or \textit{less} sensitive compared to the other information. The summary can be found in Table 3-1. In this section, we continue to examine in a more general way, of the types of communications that exist in an e-health organization. We also examine the types of information transmitted in the communication and the levels of sensitivity the information has during the communication. In this section onwards, we use the term ‘\textit{hospital}’ as an example of a health organization.

1. Types of Communications

From Table 3-1, we can see that communication in a hospital can occur over the Internet, either from within the hospital or from the hospital to the outside network. For instance, a doctor can communicate online with a nurse about a patient under their care. The doctor may also communicate with another doctor from another hospital discussing operation results. Nurses can retrieve their email using a wireless network through PDAs. The distributed nature of these communications involves processes of exchanging information that occurs through a wired or wireless network between different users.

In this section, we construct a general or typical type of a hospital environment. For simplicity purposes, the users involved in the communication either from inside the hospital local network or from the outside network are simplified and identified as Doctor, Patient, Nurse, Social Worker (SW), Paramedic, System Coordinator (SC), and System Administrator (SyA), as shown in Figure 3-1.
Seven main types of communications are identified and numbered as the following:

1. Doctor ⇔ Doctor,
2. Doctor ⇔ Patient,
3. Doctor ⇔ Nurse;
4. Nurse ⇔ Patient,
5. Paramedic ⇔ SC,
6. SW ⇔ Doctor, Nurse;
7. SyA ⇔ Doctor, Nurse, Patient, SC, Paramedic

The symbol ‘⇔’ indicates a two-way communication. The shaded area implies communication that occurs within the hospital’s local network. The communication can also occur from within the hospital to the outside network. This communication is useful particularly for users who are far away. For example, a doctor at the hospital communicates with another doctor at another hospital; a patient or SW at home communicates with a doctor at the hospital; or a paramedic at a location of an accident communicates with SC at the
hospital. The paramedic and SC work together in a distributed way. The information regarding a patient is sent by the paramedic using a PDA or a smart phone and received by SC in the hospital for further action, such as preparing for a medical team while waiting for the patient to arrive at the hospital. The public can also communicate and obtain information with the hospital, through the hospital’s website. For example, to get the hospital’s annual reports, available services, opening hours, public announcement, and information on diseases.

Table 3-2 describes the different types of information being exchanged during communication in the hospital and who is communicating it. There is information that is more sensitive than the other. For instance, information that came from communications between Doctors, Patient, Nurse, and Paramedic is more sensitive than information that came from SW⇔Nurse communications.

Doctors discuss about the critical level of a patient’s illness. A doctor discusses with a patient about his/her detailed medical information\textsuperscript{21} in a consultation session. A nurse communicates with a doctor regarding a patient’s personal information\textsuperscript{22}. The nurse also communicates with the patient, regarding his/her medication. As for SW⇔Nurse communications, only general information about a patient is involve, such as name, contact person, and award number.

\textsuperscript{21} Detailed medical information such as diagnosis, medical history, test results, current treatment, and prescriptions

\textsuperscript{22} Patient’s personal information such as name, address, age, gender, contact person, medication
**Table 3-2: Different Types of Information Exchanged between Users**

<table>
<thead>
<tr>
<th>Communication</th>
<th>Types of information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Doctor⇔Doctor</td>
<td>1. Doctors communicate with each other regarding the critical level of a patient’s illness and the best medication recommendation.</td>
</tr>
<tr>
<td>2. Doctor⇔Patient</td>
<td>2. A doctor gives consultation to a remote patient (e.g. patient at home) from hospital. Information discussed involves detailed medical information.</td>
</tr>
<tr>
<td>3. Doctor⇔Nurse</td>
<td>3. A nurse communicates with a doctor concerning a patient’s personal information, and current medical condition</td>
</tr>
<tr>
<td>5. Doctor⇔SW</td>
<td>5. A remote SW asks advise from a doctor at the hospital on a problem arises when helping a patient at home</td>
</tr>
<tr>
<td>6. SW⇔Nurse</td>
<td>6. A SW worker asks for patient’s general information from the nurse.</td>
</tr>
<tr>
<td>7. SW⇔Patient</td>
<td>7. A remote social worker communicates with a remote patient regarding appointment request for counselling sessions</td>
</tr>
<tr>
<td>Paramedic⇔SC</td>
<td>A paramedic updates patient’s information (such as patient’s personal information, medical information: allergy, blood pressure, and medical history) at a location of an accident to the database using his PDA. The information is then retrieved by SC who manages the database of accident cases</td>
</tr>
</tbody>
</table>
| SyA ⇔ All users     | Concerning user accounts  
E.g. of Scenario:  
SyA emails a nurse to collect her new user account and password at the IT department |
| Public (Open Channels) | With security- Any user that wants to get access or contact information to any sensitive information (e.g.: a researcher) |
|                     | Without security- annual reports, services available, public announcement and information on diseases |
2. Levels of sensitivity

In this section, we discuss the levels of sensitivity of the information in Table 3-2, which will be one step forward to establish our proposed MLC model. We examine the information and compare it with the levels of sensitivity already categorised in ISO 17799. We choose ISO 17799 as a comparison because the classification it proposed is well suited to our hospital environment.

If we study the communications among Doctor, Patient, and Nurse, we could find that the information that is exchanged among them includes patient’s information such as patient’s personal information, and detailed medical information. The information can be considered as extremely sensitive\(^{23}\) and should not be revealed to others except for the Patients themselves, Doctors, and the Nurses in charged.

Communications between Paramedic and SC can be considered as highly sensitive\(^{24}\), because it contains information such as data collected at the site (such as current condition of a patient, allergy types, heart rate, and blood pressure), medical history, and patient’s personal information.

Communications between Nurse⇔SW and SW⇔Patient may result in information that fall into categories between sensitive and low sensitive, which we labelled it as medium sensitive\(^{25}\). Examples of this information include patients’ information (such as name, contact person, and ward number), appointments requests, and a list of social workers that help patients either at the hospital or at home. This type of information should be treated personal and should not be disclosed to public.

\(^{23}\) We can consider the extremely sensitive to be equivalent to Top Secret category from (ISO17799)

\(^{24}\) We can consider highly sensitive to be equivalent to as Highly Confidential category from (ISO17799)

\(^{25}\) We can consider medium sensitive to be equivalent to Proprietary category from (ISO17799)
There is also information that falls into categories between low sensitive and no sensitive, which we labelled it as **low sensitive**\(^{26}\), like any non-medical related information, such as information about application systems or internal issues regarding the hospital. Although the information is not as sensitive as the other information discussed earlier, it is still considered as internal information and should not be disclosed to public. The one that can be made public is **no sensitive**\(^{27}\) information such as general information about the hospital, or general information about health, common diseases and possible treatments.

### 3.3.1.2 Classifying the Information according to the Levels of Sensitivity

In Chapter 2, (Section 2.6.1.1) we have discussed about available standards for information classification, which help organizations to classify certain information to a certain level of sensitivity, and to protect information from any unauthorized access by providing a distinguish level of security protection to the information. In this section, we classify the sensitivity levels of the information into categories based on the available standards. The reason why we categorize the information into its levels of sensitivity is that, from the categorization, we will construct our own security model (MLC in 3.3.2). In our model, we proposed suitable security mechanisms for each level of sensitivity.

In the previous section, we have identified the levels of sensitivity of the information in the hospital. In our approach, we adopt the ISO 17799 standard as our basis of information classification. We have identified all entities that contribute to the hospital’s information flow, either from within the organization or from the organization to the outside network (depicted form Figure 3-1). We have also identified the types of information that need to be protected, such as explained in Table 3-2. Now, we refer to the ISO 17799 and adopt this standard of information classification. We classify the information in e-health into five categories, together with the degree of security protection that should be applied to the information.

---

\(^{26}\) We can consider *low sensitive* to be equivalent to *Internal Use Only* category from (ISO17799)

\(^{27}\) We can consider *no sensitive* to be equivalent to *Public Documents* category from (ISO17799)
i. Top Secret: contains extremely sensitive patient’s information. The distribution of this kind of information is very restricted and must be protected all the time. Highest security protection must be applied. Examples of such information are patient’s personal information\textsuperscript{28} and detailed medical information\textsuperscript{29}.

ii. Highly Confidential: contains highly sensitive information, related to the patient’s information that should not be shared internally or made public. It includes information that is obtained from mobile devices to the organization. For example, patient’s personal information, and medical information (such as allergic, blood pressure, and medical history); which is sent from a remote paramedic at an accident spot. Security should be very high and suitable for devices with limited resources.

iii. Proprietary: contains medium sensitive information related to the information that is required for the operational work routines of the hospital’s staff. Examples of the information include patient’s information (such as name, contact person, and ward number), general medical information (which is not involved detailed medical information). Use by authorized personnel only. Security at this level is medium high.

iv. Internal Use Only: contains low sensitive information, which is not approved for general circulation outside the organization. Examples of the information are information on application systems, user accounts, and non-medical related information such as computer technical problems. Security at this level is low.

v. Public: Information that can be disclosed to public. Security at this level is minimal.

In this classification, we choose to use highest, very high, medium high, low, and minimal to distinguish the degree of security or the security level provided in each categories\textsuperscript{30}.

3.3.2 Analysis

In this section, we introduce our MLC model which is to (1) classify the communications in e-health according to the levels of sensitivity of the information transmitted, and (2) proposed appropriate security mechanism to secure the communications. The MLC classifies

\textsuperscript{28} Patient’s personal information, Refer footnote 22

\textsuperscript{29} Detailed medical information, Refer footnote 21

\textsuperscript{30} This term is equivalent to the term used in ISO 17799 (Section 2.6.6.1), which are highest, very high, high, controlled but normal, and minimal.
communication into five layers, and provides five categories of security mechanisms for the communication. In our approach, we do not focus on how, or in what way the information is stored or accessed. Our approach centred on how to secure communication sessions between two points, which transmit information that has different levels of sensitivity.

3.3.2.1 MLC: Classifying the Communication

We are interested on how to classify every communication between users in e-health, based on the levels of sensitivity of the information transmitted during the communication. By classifying the communication, we can provide flexible security mechanisms around the communication based on organizational needs.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Sensitivity of the data</th>
<th>Types of data communicated</th>
<th>Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1</td>
<td>Top Secret</td>
<td>Contains Extremely Sensitive information: Patient’s personal information and detailed medical information</td>
<td>Doctor⇔Doctor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Doctor⇔Patient</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Doctor⇔Nurse</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Nurse⇔Patient</td>
</tr>
<tr>
<td>Layer 2</td>
<td>Highly Confidential</td>
<td>Contains Highly Sensitive information: Patient information that should not be shared internally or made public, and information obtained from the paramedic at an accident spot</td>
<td>Paramedic⇔SC</td>
</tr>
<tr>
<td>Layer 3</td>
<td>Proprietary</td>
<td>Contains Medium sensitive information: Patient’s information that is required for the operational work routines of the hospital’s staff.</td>
<td>Doctor⇔SW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Nurse⇔SW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Patient⇔SW</td>
</tr>
<tr>
<td>Layer 4</td>
<td>Internal Use Only</td>
<td>Contains Low sensitive information: Any information that is not approved for general circulation outside the organization.</td>
<td>SyA ⇔ all users</td>
</tr>
<tr>
<td>Layer 5</td>
<td>Public</td>
<td>Open channel: No sensitive information such as general information on the hospital, information on health, diseases, frequently asked questions, annual reports, and services available</td>
<td>The public</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Secure open channel: any user, e.g. a researcher who wants to get access or contact information to any anonymous sensitive information (Austin &amp; Lemmens, 2009; IOM, 2000).</td>
<td></td>
</tr>
</tbody>
</table>
We propose the communications in e-health to be categorized into five layers, which is Layer 1 to Layer 5, based on the five classifications of information described in Section 3.3.1.2. Table 3-3 shows the five types of communications in MLC. The communication in Layer 1 contains data with the highest level of sensitivity (Top Secret), while Layer 5 contains data with the lowest level of sensitivity (Public). The security protection suggested in each layer is in accordance with the security provided in the ISO 17799:

**Layer 1:** For communication between users that exchange Top Secret information, which is extremely sensitive. The **Highest** protection mechanisms should be applied. The information should be protected against threats and loss, and disclosed only to authorized users such as doctors, patients themselves and the nurses in charged. Any disclosure to other users must follow the patients’ consent.

**Layer 2:** For communication between users that exchange Highly Confidential information, which cannot be shared internally or made public. This includes information which is obtained from **mobile devices** to the organization. Security at this layer should be **very high** and suitable for devices with limited resources.

**Layer 3:** For communication between users that exchange Proprietary information, which is required for the operational work routines of the hospital. Security at this level is **medium high**.

**Layer 4:** For communication between users that exchange Internal Use Only information, which is related to general information about the organization’s system and non-medical related information. Security at this level is **low**.

**Layer 5:** For communication between users that exchange Public information. This layer is divided into two that are with security, called secure open channel, and without security, called public open channel. Security at this level is **minimal**.

So far, we have proposed the communication classification, and explained what types of sensitive information are transmitted at each layer. We also explained to what level the security must be applied. In the next section we discuss how we proposed protection mechanisms at each layer with different levels of security, by using cryptography protocols.
3.3.2.1 Security Requirements for a secure e-health communication environment

Before we proposed our security mechanisms, we analyse the security requirements needed to create a secure e-health system. Security requirements are derived from organizational needs and different from one organization to another. According to Bleumer (1994), the requirements for e-health systems are authentication to enter the system, confidentiality to prevent disclosure of the information, and anonymity (for example, to get consultation from expert system anonymously). Rodriguez et al. (2006) added the requirements as integrity on data storage, access control over the protected health information, non-repudiation to all information flows, attacks detections, as well as security auditing. Blobel and Roger-France (2001) created a general security model based on the communication aspects of an application. The elements required to secure the communication were identification, authentication, access control, integrity, confidentiality, and availability.

Although different organizations have different set of security requirements according to their needs, however, basic security requirements must concentrate on addressing the aspects of confidentiality, integrity, and availability (Pfleeger & Pfleeger, 2006). In this research, our focus is to secure the process of message exchanges between two points, which is between a sender and a recipient in different communicating environments. Both of the users would want to make sure that the message sent or received is safe from any unauthorized access (confidentiality), not modified (integrity), and the originality of the message is guaranteed (non-repudiation).

The sender would also want to make sure that he/she can prove that the message is from him/her (non-repudiation). The recipient would want to make sure that he/she can access the message whenever he/she needs to (availability). These requirements often addressed using cryptography (Ametller et al., 2004b; Foner, 1997; Mehta et al., 2007; Ugurlu & Erdogan, 2005). We refer to Chapter 2, Section 2.3.3, for the cryptography protocols that are encryption, decryption, hash function, digital signature, and digital certificates, to address the following security requirements:
1. Confidentiality is often associated with the encryption/decryption of a plaintext. An encryption transforms the plaintext into a ciphertext (and vice versa for decryption) using a shared symmetric key, in such a way that a third party cannot recover the actual text from the plaintext. By ensuring that only the intended recipient can recover the plaintext, availability is also included.

2. In order for the sender to make sure that the plaintext is not modified, a hash message can be computed from the plaintext and embedded into the message before sending it to the recipient. The recipient can verify the hash message by recalculating a new hash message from the plaintext.

3. The sender can provide a proof that he/she sends the message by using a digital signature, by encrypting parts of the message with his/her private key, so that the recipient can verify it using the sender’s public key.

4. The recipient(s) public key can be handled systematically by using digital certificates that bind a user with his/her public key.

Based on these requirements, we design our proposed secure communication model for each layer in MLC, which can provide flexible security features.

**3.3.2.2 MLC: The proposed security protection mechanisms**

As discussed in Chapter 2 (Section 2.6.1.1), the more sensitive data has greater degree of loss or potential damage compared to the less sensitive data. It is a common practice for medical staff to keep patient’s information a private while in transit, which is stated in several standards such as in (Privacy-Act, 1974), HIPPA (http://www.hhs.gov/ocr/hipaa/), and APA (http://www.privacy.gov.au/). However, these standards do not include how the implementation of security mechanism should take place to protect the information.

The proposed security protection mechanisms in MLC are based on the security requirements explained in the Section 3.3.2.1, which use cryptography protocols. The MLC is taking into account of providing flexible security protections in order to address security needs in e-health. The MLC provides three types of security mechanisms, which are data security, channel security, as well as data and channel security. Data security uses cryptography
protocols such as symmetric encryption/decryption, hash function, and digital signature, while channel security uses the SSL protocol. We discuss each of the MLC’s security mechanisms in details in the following sections.

**Mechanism 1: Data security**

Consider two communicating points (a sender and a recipient) depicted in Figure 3-2. The sender wants to send a plaintext to the recipient. Both of them need cryptography protocols to secure (and recover) the plaintext.

The following describes the notations used in the cryptography processes:

- Public and Private keys of the recipient \((pubKr, privKr)\)
- Public and Private keys of the sender: \((pubKs, privKs)\)
- Symmetric keys \(K\);
- Plaintext, \(P\), Hash of Plaintext, \(H(P)\)
- Digital signature, \(S\)
In our approach, we use the symmetric key encryption, hash function and digital signature to provide data security. The following describes the step-by-step process at the sender’s and recipient’s sides:

a. **Cryptography Protocol at the sender side**

i. *Symmetric encryption*: encrypts the plaintext into ciphertext using a key $K$. The encryption process ensures the confidentiality of the plaintext.

$$\text{Ciphertext} = E(P)K$$

ii. *Hash function*: computes hash value from the plaintext, $H(P)$. The hash value will be used by the recipient to check the integrity of the plaintext, and verify whether the plaintext is tampered. The recipient recalculates the hash value from the plaintext retrieved from the ciphertext and compares it to the one sent by the sender. If both are matched, then the plaintext is genuine and the integrity of the plaintext is verified.

iii. *Key exchange*: the key $K$, should be encrypted and sent to the recipient, so that $K$ can be used to decrypt the message at the recipient’s side. In order for the sender to make sure only the recipient can recover the key, $K$ will be encrypted with the recipient’s public key, $pubKr$. To avoid a third party to steal and remove $H(P)$ that is computed earlier, it can be encrypted together with $K$ using $pubKr$, and we name the result of the encryption as *Cipherkey*.

$$\text{Cipherkey: } E(K, H(P)) pubKr$$

iv. *Digital Signature*: in order for the sender to prove that the *Cipherkey* is from him/her, the sender signs it using his/her private key ($privKs$) to produce signature $S$.

$$S = E(\text{Cipherkey}) privKs$$

v. *Send message*: Afterwards, the sender can send Ciphertext, Cipherkey, and $S$ to the recipient. In our approach, we use HTTP protocol to transfer message for the wired network, so that SSL can be used to secure the channel. For the wireless network, we use the Global System for Mobile communications (GSM) network, or wireless LAN (WiFi) to transfer the message.
b. **Cryptography Protocol at the recipient side:**

i. To check that Cipherkey is indeed come from the sender, $S$ is verified against Cipherkey

ii. If Cipherkey is valid, then the following is executed:
   a. use $privKr$ to decrypt Cipherkey
      
      $D(Cipherkey) \cdot privKr = K, H(P)$

   b. Then, use $K$ to decrypt Ciphertext
      
      $D(Ciphertext)K = P$

   c. Finally, verifies $P$ by calculating a new $H(P)$ from $P$, and compare it with the one in (a). If proved valid, keep $P$.

These basic steps of data security will be used and expanded later in Chapter 5 (Section 5.3.4), to tailor or adapt with the use of the agent approach.

**Mechanism 2: Channel security**

In the channel security, the sender and recipient exchanges certificates, and then the ender establishes SSL channel to the recipient side, and simply transfer the plaintext. Certificates can be obtained through the Security Administrator in an organization, which is in charged with creating identification (Id) and a password for user accounts.

**Mechanism 3: Both data and channel security**

When using option of both data and channel security, Sender sends all Ciphertext, Cipherkey, and Signature $S$ to the recipient through the SSL channel.
3.3.2.3 The Key size for the symmetric key encryption

The key $K$ is an important component of an encryption process because it represents the level of security that the algorithm can provide. According to Lenstra (2004), a symmetric cryptography system with $n$-bit of keys has a security level of $n$, if it can endure a generic attack (to find the key, when plaintext and ciphertext are known beforehand), using efforts less than the exhaustive search or ‘brute-force’ attack. The selection of the key size is based on the level of security required for a cryptography system. The longer the key, the higher the security it can provide because the difficulty of trying all possible keys in the exhaustive search is directly proportional to the number of bits used (Blaze et al., 1996). This answers why shorter key sizes can only provide low security as it will take less time to find the key using the exhaustive search, compared to longer key sizes.

The MLC model provides different symmetric key sizes for each layer. Layer 1, which protects the Top Secret information, should provide the highest security available and therefore, the strongest key must be provided. Layer 2, which protect the Highly Confidential information should be able to provide very high security, Layer 3, which protects the Proprietary information, should provide high security, and Layer 4, which protects the Internal Use Only information, should provide low security.

The US government policy provides recommendations on the symmetric key sizes to protect classified information namely Top Secret, Secret, and Confidential information (CNSS, 2003). The Advanced Encryption Standard or AES algorithm (FIPS197, 2001) is chosen for this purpose. AES-192 bit or AES-256 bit is chosen to secure the Top Secret information, while AES-128 bit is chosen to secure both Secret and Confidential information.

Debates on selecting symmetric key sizes for a cryptography system has been and still going on. The selection process is often based on the amount of time it will take for an attacker to attack the key and how much of resources (cost) one needs to succeed. For example, one can spend $400 on a Field Programmable Gate Array (FPGA) machine to find 40-bit key within 5 hours (Blaze et al., 1996). It is important to make sure that the key size chosen for a cryptography system is proven to be strong. There are many efforts to find flaws in the key
size for certain algorithms mainly using brute-force. Brute-force attacks can be achieved by computing in parallel that is, one can easily add as many processors as desired to perform partial search of the key. Apart from the brute-force attack, there are also two other attacks to find the key including attack using specific machines (such as Application-Specific Integrated Circuit (ASIC) machine and FPGA Machine); and the Time-Memory-Data Trade-offs attack, that is a condition where the attacker is able to pre-compute and generate the encrypted data using random keys beforehand and store in a database. Then, the attacker observes the actual encrypted data and sees if he/she can find a match pattern between the randomly generated data and the actual data. The summary of the amount of time and cost required for the attacks are explained in details in (ECRYPT, 2008). The RSA Laboratories organize cryptographic challenges, which offer prizes for any individual who can break a specific key (the details can be found in http://www.rsa.com/rsalabs/node.asp?id=2091). The 56-bit DES key has been successfully cracked by the Electronic Frontier Foundation (EFF) in 1999. They used equipments worth of $250,000 that use 100,000 computers to find 56-bit key in 22 hours (EFF, 1998).

These attack efforts are the motivation for many researchers to determine the appropriate symmetric key sizes for a cryptographic system. Many suggestions have been made regarding the selection of the symmetric key sizes selection. ECRYPT (2008) argued that different information has different lifespan, and a key size selected to protect a particular information should be larger than the lifespan of the information. For examples, electronic banking transactions have brief security protection and private information like medical information needs protection for a lifetime of a patient.

In the late 1995, Blaze et al. (1996) made an ad-hoc report regarding the minimum symmetric key sizes required for commercial security. The report was made to discuss a solution and address the problem of inadequacy of the confidentiality protection provided by the existing key sizes. They reported that a symmetric cipher with 40-bit key does not provide any protection against brute force attack, and added that the 56-bit of DES (Mehuron, 1999) is considered inadequate, although Lenstra (2004) argued that there was not any attack that could break DES with security level of 56, except for the exhaustive search of the key. Blaze et al.
suggested that 75-bit key was adequate in the late 1995 based on the available equipments and
time needed to find 40-bits and 56-bits keys at that time. They then proposed that 90-bit key
was the minimum key size required to provide security for the next 20 years (from late 1995).
ECRYPT (2008) supported Blaze et al.’s report, and claimed that the method is still
reasonable to be exercised.

Lenstra (2004) came out with a formal formulation on how to determine key sizes for
symmetric key with the lifespan of the key. This formulation was an updated version of his
works in 2000 (Arjen & Eric, 2000). Lenstra’s work was based on the DES 56-bit key, which
was first introduced in 1977. DES was first being reviewed in the year of 1982. He suggested
that DES has provided *adequate protection* in the year of 1982. Based on this, he studied the
next security level required in proportion with years. He referred to the Moore’s Law which
was formulated in 1965 (Fibíková & Vyskoc, 2001; Lenstra & Verheul, 2000) stating that the
amount of computing power and random access memory one gets, doubles every 18 months.
He then suggested that the security level should also be increased by one for every 18 months,
starting from year 1982. For example, a cryptography system should use 66-bit (56-bit + 10)
in 10 period of 18 months (which is equivalent to 15 years), and therefore should give
adequate protection in 1997 (which is obtained from 1982 + 15).

The following equation was introduced by Lenstra (2004) to find the adequate key size, \( K \) in
year \( Y \):

\[
K = 56 + 2 \left( Y - 1982 \right) / 3
\]  

(1)

For example, in 20 years time from 2009, (which is 2029) the adequate key size is \( K = 56 + 2 
\left( 2029 - 1982 \right) / 3 = 87 \), in other word, 87-bit keys should be used until the year of 2029 to
provide adequate protection. We can also find \( Y \), if given the key size \( K \) by:

\[
Y = 1982 + 3 \left( K - 56 \right) / 2
\]  

(2)

Based on (Blaze et al., 1996) and (Lenstra, 2004)’s works, ECRYPT (2008) recommended key
sizes with the lifespan of the key, shown in the Table 3-4.
Table 3-4: Security levels excerpt from Table 7.4 from ECRYPT (2008)

<table>
<thead>
<tr>
<th>Security (bits)</th>
<th>Protections</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>Very short-term protection against agencies,</td>
<td>≤ 4 years protection</td>
</tr>
<tr>
<td></td>
<td>long term protection against small organizations</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>Legacy standard level</td>
<td>≈ 10 years protection</td>
</tr>
<tr>
<td>112</td>
<td>Medium-term protection</td>
<td>≈ 20 years protection</td>
</tr>
<tr>
<td>128</td>
<td>Long-term protection</td>
<td>≈ 30 years protections</td>
</tr>
<tr>
<td>256</td>
<td>“Foreseeable future”</td>
<td>Good protection against quantum computers (Shor, 1997)</td>
</tr>
</tbody>
</table>

ECRYPT (2008) reported that 80-bit key is suitable for a very short term protection against a brute-force attack, and added that if an attacker is able to pre-compute the data (as in the Time-Memory-Data Trade-offs attack), the 80-bit key is breakable. The report also stated that the 32 and 64-bit keys are not suitable for confidentiality protection because the 32-bit key does not offer any protection, while the 64-bit key offers very poor protection.

Table 3-5: The existing key size recommendations

<table>
<thead>
<tr>
<th>Recommended key size (in bit)</th>
<th>75</th>
<th>80</th>
<th>90</th>
<th>96</th>
<th>112</th>
<th>128</th>
<th>192</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blaze</td>
<td>Adequate until late 1995</td>
<td>Adequate until 2015 ≈ 20 years</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECRYPT</td>
<td>≤ 4 years</td>
<td>≈ 10 years</td>
<td>≈ 20 years</td>
<td>≈ 30 years</td>
<td>Foreseeable future</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>US Policy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Confidenti al &amp; Secret</td>
<td>Top Secret</td>
</tr>
<tr>
<td>Lifespan&lt;sup&gt;31&lt;/sup&gt;</td>
<td>≈3 years</td>
<td>≈ 10 years</td>
<td>≈ 25 years</td>
<td>≈ 34 years</td>
<td>≈ 58 years</td>
<td>≈ 82 years</td>
<td>≈ 178 years</td>
<td>≈ 274 years</td>
</tr>
</tbody>
</table>

---

<sup>31</sup> The calculations are from year 2008, based on Lenstra (2004)
We calculate the lifespan for each key length using Lenstra (2004)’s formulation in (2) shown in Table 3-5 in the last row. We compare the duration of protection given by (Lenstra, 2004) with (Blaze et al., 1996), (ECRYPT, 2008), and the US Policy (CNSS, 2003).

Although there is a huge gap of lifespan between ECRYPT and Lenstra formulations, we can summarize that both recommendations, as well as the US policy suggest:

1. 256-bit key and 192-bit key provide **highest** security for a very long term protection,
2. 128-bit provides **medium high** security for a long term protection,
3. 112-bit provides **medium** security for a medium term protection, and
4. key bits from 80-bit provides **low** security for a short term protection.

From the summary, we recommend the symmetric key sizes value for every layer in the MLC model is provided in *ranges* like the following:

- **193-bit and longer:** suitable for Layer 1, to secure the Top Secret information that needs the **highest** security protection
- **129-bit to 192-bit:** suitable for Layer 2, to secure the Highly Confidential information that needs a **very high** security protection
- **112-bit to 128-bit:** suitable for Layer 3, to secure the Proprietary information that needs a **medium high** security protection
- **80-bit to 111-bit:** suitable for Layer 4, to secure the Internal Use Only information that needs a **low** security protection

Table 3-6 describes the recommended key sizes in each layer in MLC. The US Policy recommendation is also included for comparison purposes.

<table>
<thead>
<tr>
<th>US Policy</th>
<th>Key lengths (in bit)</th>
<th>MLC</th>
<th>Key lengths (in bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top Secret</td>
<td>192/256</td>
<td>Layer 1 (Top Secret)</td>
<td>193 and longer</td>
</tr>
<tr>
<td>Secret</td>
<td>128</td>
<td>Layer 2 (Highly Confidential)</td>
<td>Wired: 129-192 Lightweight device: 112-192</td>
</tr>
</tbody>
</table>
The table shows that Layer 1 and Layer 2 key sizes are aligned with the US’ Top Secret key sizes (192-bit for Layer 2, 193-bit and longer for Layer 1). Layer 2 supports mobile devices security, and therefore, key length as low as 112-bit is supported for low processing power device. For Layer 3, we choose 112 to 128-bit key to provide medium security, which also aligned with US’s Secret key sizes. For Layer 4, key sized from 80-bits to 111-bit are chosen to provide low security. By providing key length values in certain ranges, we can offer a wider range of key sizes for each layer.

In summary, we conclude the security mechanism in the MLC model, which includes data and channel security as depicted in Table 3-7.

<table>
<thead>
<tr>
<th>Layers</th>
<th>Security Mechanisms</th>
<th>Key lengths in bit for data security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1 (Top Secret)</td>
<td>Data and channel security</td>
<td>193 and longer</td>
</tr>
<tr>
<td>Layer 2 (Highly Confidential)</td>
<td>Data or channel security *mobile devices use data security only</td>
<td>Wired: 129-192 Wireless:112-192</td>
</tr>
<tr>
<td>Layer 3 (Propriety)</td>
<td>Data or channel security</td>
<td>112-128</td>
</tr>
<tr>
<td>Layer 4 (Internal Use Only)</td>
<td>Data or channel security</td>
<td>80-111</td>
</tr>
<tr>
<td>Layer 5 (Public)</td>
<td>-</td>
<td>ID and Password (for secure open channel)</td>
</tr>
</tbody>
</table>

For channel security, ciphersuites from any available provider can be used to provide protection. For example, the SunX509 provides the following ciphersuites shown in Table 3-8:
### Table 3-8: Ciphersuites provided by SunX509 provide

<table>
<thead>
<tr>
<th>Bit</th>
<th>Ciphersuites</th>
</tr>
</thead>
<tbody>
<tr>
<td>256-bit</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA, TLS_DHE_RSA_WITH_AES_256_CBC_SHA</td>
</tr>
<tr>
<td>168-bit</td>
<td>SSL_RSA_WITH_3DES_EDE_CBC_SHA, SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA</td>
</tr>
<tr>
<td>128-bit</td>
<td>SSL_RSA_WITH_RC4_128_MD5, SSL_RSA_WITH_RC4_128_SHA, TLS_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_RSA_WITH_AES_128_CBC_SHA</td>
</tr>
</tbody>
</table>

Because of the limitation of the available ciphersuite provided from the SSL providers, (which only provides Layer 1 with 256-bit, and Layer 2 with 168/128-bit protection), we could use 128-bit ciphersuites as alternatives for Layer 3 and Layer 4 as well.

For Layer 1, *data and channel* security are used to provide the highest protection mechanism. The key lengths for data encryption are from 193-bit and above. Layer 2 uses data or channel security only. For data security, 129-bit to 192-bit of keys are used with the wired network, while 112-bit to 192-bit of keys are chosen for the wireless network. Layer 3 and Layer 4 also provide two options either data or channel security, with key lengths of 112 to 128-bit and 80 to 111-bit respectively. For Layer 5 that is intended for public use, we could use ID and password only, to support secure open channel, which is discussed in Table 3-3.

#### 3.3.2.4 MLC Model: Justifications and Advantages

Online communications play a major role in e-health, to disseminate and exchange information among users such as doctors, nurses, patients or social workers. Different users communicate different types of information. There is sensitive information that has to be kept confidential, and there is also information that can be shared with public. With the pervasive use of the Internet, information can be accessed anywhere regardless of the geographical location or economic status because the Internet offers cheap and worldwide coverage. Remote users such as patients can now use the Internet to communicate with their doctors and nurses from home and be part of e-health users.
The MLC model provides security mechanisms to secure different types of communications among different users in e-health according to their needs. For example, a nurse can communicate through a communication, which is secure or less secure depending on the situation. The nurse can communicate through the highest level of security when communicating with doctors or patients. Alternatively, he/she can use a medium level of security when communicating with SWs, or a minimum level of security when communicating with SA\textsuperscript{32}.

By using different combinations of key sizes for data and channel security, flexibility of providing security can be provided to the health organizations. Different security strengths can be provided at each layer depending on the sensitivity of the data. The extremely sensitive information can be secured using the highest security mechanisms, while low sensitive information can be secured with minimal security mechanisms. Therefore, any excess security applied on the communication can be avoided when it is not needed. MLC satisfies the current technologies gaps and limitations, such as discussed in Section 2.6. By using MLC, users are now able to communicate with different types of security mechanisms suitable for their needs.

A set of encryption algorithms that are proven to be reliable by experts can be chosen to secure the layers. The selection of the algorithms can be made or decided by the Security Administrator in a particular organization. In MLC, there are data and channel security provided to users in such a way that the user can choose the most suitable security processes in terms of cost and efficiency. For example, the organization can choose SSL channel for the communication, which is cheaper than the data encryption (French, 2006), however, with a trade-off of inflexible security configuration when the user needs to change to stronger or weaker security level. Alternatively, the organization can choose to use data security only, with suitable encryption key sizes such as described in Table 3-7. Meanwhile, when especially excess security is needed for an extremely important communication, the organization can opt for data and channel security.

\textsuperscript{32} Refer to Table 3-3
In addition, communication with low processing power devices, such as PDAs and smartphones are provided with appropriate data security with key sizes available from 112-bit key. The organization can save resources such as CPU processing power for the lightweight devices using appropriate key lengths to give better performance to the communication.

However, there is always a trade-off between strong security and performance. The longer the key lengths, the slower the performance of the security processes. Longer key length provides better security because more works and efforts are required by the attackers to find the key. Therefore, if security is important, one may select stronger algorithms with decreasing performance. Otherwise one may choose shorter key lengths with high performance according to the needs.

We have so far proposed the MLC model with five layers of communications based on the classification of the sensitive information. We also have justified the need for multilayer security. Next, we will examine our proposed solution models to implement the MLC model.

3.4 Solution Models

So far, we have identified the problem characteristics, and analyse the requirements and possible solutions. We arrived at the MLC model as a solution to cater for the communication problem in e-health. In order to realize the MLC model, we adopt the Software Engineering approach to design and develop, to implement, as well as to test and evaluate the MLC model.

3.4.1 Design of Solutions

We develop a security system, which is based on MAS, to implement the security mechanisms proposed in MLC. We use MAS to handle the different types of user requirements and the different types of security processes. We also develop a traditional, non-agent based system for comparison purposes.

In the system based on MAS, we model the agents so that they are skilled or knowledgeable to cater for the security processes defined in MLC, and we called the model as the Multi-agent based Security Model (MAgSeM). MAgSeM applies MLC and uses collaborative agents to
solve a goal, which is to communicate and exchange messages between a sender and a recipient. Chapter 5 will discuss MAgSeM in greater details. For the traditional non-agent based system, we develop a system which has similar functionalities of sending and receiving a message.

### 3.4.2 Implementation

We implement the MAgSeM-based system and the traditional non-agent based system over the TCP/IP network. The MAgSeM-based system is implemented using JADE (Bellifemine et al., 2001; Bellifemine et al., 2007), which is based on Java (Arnold & Gosling, 1998), while for the traditional non-agent based system, we use the socket programming in Java. Both systems are implemented in such a way that they could support two-way communication between a sender and a recipient.

We also include implementations for MAgSeM-based and non-agent based systems on lightweight devices. We use Java 2 Platform Micro Edition (J2ME) platform to develop systems in mobile devices (http://java.sun.com/javame/index.jsp). We use J2ME together with Connected Limited Device Configuration (CLDC) and The Mobile Information Device Profile (MIDP), a complete set of requirements in order to develop applications, which is also known as MIDlets (Goyal, 2005). Details of the implementation are discussed in Chapter 6.

### 3.4.3 Test and evaluation

We carry out experiments, which compare the execution times for the agent-based and non-agent based systems. We compare both systems in terms of the following:

- a. The communication performance for both systems at every layer, which uses different security mechanisms, on the wired machines and wireless devices
- b. The security overhead imposed by both systems
- c. How agents provide better control over the security processes compared with the traditional non-agent based system, and simultaneously narrows the gaps on current security technologies.

We discuss the result and provide analysis from the experiments in Chapter 7.
3.5 Summary

In this chapter, we have presented our proposed MLC that will be our main foundation in this thesis. We have suggested the information classification, which is adopted from the ISO 17799. Based on this classification, we classify the communications in e-health into five layers. In each layer we proposed suitable security mechanisms that cover data and channel securities. Relevant key lengths are also proposed in each layer to represent the security level provided to the information.

But, before we go further on the designing, implementing, testing and evaluating the systems, we examine the characteristics and the advantages of MAS approach that is used to model the security processes, and how it is suitable for distributed problem such as communications in e-health, which will be discussed in Chapter 4.
CHAPTER 4  MODELLING TRADITIONAL APPROACHES THROUGH MULTI-AGENT SYSTEM

4.1  Introduction

In Chapter 3, we have presented our MLC model that provides flexible security mechanisms characterized by five layers to secure online communication in the e-health domain. We also have discussed the possibility to use the MAS approach to cater for the security processes in the MLC model. This chapter investigates how MAS supports the traditional non-agent based systems and studies how MAS is suited to be used in our proposed model. In Section 4.2, we first discuss the weaknesses or drawbacks of traditional systems and study why MAS is chosen to support them. Then we identify processes to secure the online communications that use the MLC model as security mechanisms, and examine how MAS is suited to cater for these processes. In Section 4.3, we pinpoint the desired characteristics of the agents, and study these characteristics in further details. Lastly we justify why MAS is chosen for designing and implementing our security model.

4.2  MAS Characteristics Supporting Traditional System Approach

4.2.1  Inadequacies of traditional approaches

Since the introduction of the agent technology more than a decade ago, the technology has been a supporting tool for many applications\(^\text{33}\). In Chapter 2, we have seen MAS examples in applications such as in the information retrieval, air traffic control, electronic commerce, business process management, medical domain, personal agent, and security field. In this section, we discuss about the weaknesses of the traditional non-agent based systems approach and how this leads to the selection of MAS for the solutions. According to Jennings (2000),

\(^{33}\) Refer to Section 2.5.4.1.
traditional systems have two main weaknesses especially when it comes to developing complex and distributed systems, which are (1) the components interactions are defined in a rigid way, and (2) mechanisms to represent the system’s organizational structure are insufficient.

We choose to discuss the drawbacks of the traditional non-agent based systems such as the Knowledge Management System (KMS), Legacy Information System (LIS), and Groupware and Computer Supported Cooperative Work (CSCW), and then discuss how the MAS approach support these systems. The similarities among these systems are that they may involve distributed environments and complex systems.

Knowledge Management System (KMS) is a system that allows organizations to use and manage their information and knowledge to better run their organization operations. According to Nabeth et al. (2003), a smart KMS is not only able to search and store knowledge, it should also be able to create, transform, manipulate, communicate, share, assimilate, and apply that particular knowledge.

Nabeth et al. argued that current KMSs have limitations such as (1) unable to support the continuous, active and dynamics management of knowledge of the users: current KMS only able to support responds to user’s queries in the form of tree-like categorizations of static documents, although it provides some sophisticated searching algorithms. They argued that KMS should be able to provide advanced assistance and guidance throughout the searching processes, and they should be delivered in “a richer and livelier” forms compared to the static documents; (2) have limitation in managing tacit knowledge: because of the dynamic nature of the organization and often intangible factors are involved in the organization (such as experience, skills, practices, and know-how), tacit knowledge is often difficult to manage; and (3) unable to support user interactions and user preferences. KMS should be user centric where all users preferences must be taken into account to support efficient activities of the users, deliver knowledge according to their preferences, and able to motivate them. Based on these weaknesses, Nabeth et al. proposed a KMS that is based on MAS. To support user interactions based on user preferences, an agent is designed to assist a user by monitoring
user’s actions and give assistance to the user. A cognitive agent (Wooldridge & Jennings, 1995a) is used to learn the user’s behaviours and preferences, deliver knowledge to the user based on their preferences, stimulate the user in the form of providing alternatives and questions, and guide users to use the knowledge to its highest potential.

Zhang et al. (2008) suggested that centralized KMS is not able to cater for or adapt to distributed environment. They designed a KMS that is based on MAS, because of the nature of the agents as distributed problem solvers, where the agents are able to use distributed information and distributed expertise, in a distributed environment (Sycara, 1998), the MAS approach is suitable to solve this problem. Each agent is specialized with certain skills that represent processes in KMS. These agents have different goals and state and therefore, they must coordinate their actions in order to attain their own goals. Other examples of KMSs that are based on MAS can be found in (Dai et al., 2003; Lee & Lee, 2005; Tacla & Barthes, 2002).

The Legacy Information Systems (LIS), which can be defined as “any information system that significantly resists modification and evolution” (Brodie & Stonebraker, 1995) has the following drawbacks Bisbal et al. (Bisbal et al., 1999) and Bisbal et al. (1997): (1) LIS run on obsolete hardware, which has costly maintenance and slow in performance, (2) software maintenance are costly due to lack of documentation and time consuming, (3) difficult to be integrated with other systems because of no clean interface with other system, and (4) LIS is difficult to expand. On the other hand, MAS has standardization that makes it easily be integrated with other system. The Foundation for Intelligent Physical Agents (FIPA) is an organization that provides agent development standards and specifications, including agent communications and agent management (Bellifemine et al., 2007). In addition, MAS permits interconnection/interoperation of multiple legacy systems, that is, to integrate the legacy systems with MAS, so that the systems can utilize the agents capabilities (Sycara, 1998). In addition, an agent in MAS is extensible, where it can easily be instantiated with new functionalities and integrated to the existing system Debenham (1999) and Sycara (1998). Therefore, adding and removing agents can be performed without having to reconfigure the
whole system. Nguyen et al. (2008) and Zgaya & Hammadi (2006) provide examples of information systems with support from MAS.

Groupware and Computer Supported Cooperative Work (CSCW) are software and hardware systems that assist users, which are working in group discussions or meetings. However, current approach does not support flexible user interactions and it does not able to model highly dynamic and complex processes where the knowledge of the domain is incomplete (Bergenti et al., 2002; Rong & Liu, 2006). In order to support flexible interactions in the group discussion, agents are used to represent users in online collaborations between remote and mobile user (Bergenti et al., 2002). A Personal Agent acts on behalf of a participant to assist in automating a negotiation or a decision making process, scheduling a meeting, and providing event notifications to users. A Session Manager Agent acts as a facilitator to every group discussion and provides access to resources. Agents are also used to perform supporting tasks within a group discussion. Ellis and Wainer (1999) demonstrated the team-agent to support CSCW system. The team-agent only concern on a general functionality of the system rather than the main content of the discussion, such as “performance specialist” within a software engineering team, and the “social mediator” within an electronic meeting. There are three types of team-agents that are autonomous agents, single user agents, and group agents. Autonomous agents are responsible for independent subtasks. Single user agents such as Interface Agent are responsible to interact and act on behalf of a participant. Group agents are responsible for interacting and collaborating with other agents that represent other participants.

Agents can also be used to support the highly dynamic and complex processes through their autonomy, reactivity, and social abilities characteristics. Rong & Liu (2006) designed a MAS model to support CSCW system modelling, based on semiotics perspective. In this model, CSCW is viewed as a social model of human behaviour in the form of human agents.

From the above discussion, we investigate how MAS is used as supporting tools to enhance the traditional non-agent based system using the agents’ characteristics. In the next section, we investigate the suitability of MAS to MLC.
4.2.2 MAS for MLC

As discussed in Chapter 3, the purpose of the MLC model is to develop a flexible and secure communications system for domains like e-health. In this section, we will discuss the suitability of the MAS’s properties to support the processes in the MLC model. First we identify the processes of creating a secure communication and then we identify how MAS is suitable to represent the processes.

The main processes to develop a secure communication system, which applies the MLC model, are like the following:

1. Determine which layer the communication between a sender and a recipient is in, at the MLC model so that appropriate security mechanisms can be chosen.
2. If more than one communications are required by the sender, the layer for each communication must be determined, and for each communication, the security processes must be catered for separately
3. Perform cryptography protocols to prepare the data at the sender’s side, such as in the Section 3.3.2.2.
4. Send a request to send a message to the intended recipient
5. If the recipient agree, send to the secure message
6. Perform cryptography protocols to recover original data at the recipient’s side
7. If the communication is to a recipient on a mobile device, perform security processes suitable for mobile device

These tasks can be handled by the multi-agent system, because:

1. The agents can represent a user to handle security processes automatically. The security processes are done stage by stage and distributed because they needs resources from remote recipient such as permission to send a message as well as the private key of the recipient when decrypting Cipherkey (Section 3.3.2.2) to recover the plaintext.
2. The autonomy and interactive characteristics allows agents with different capabilities to secure the data, which involve interacting with the user/sender, organizing information obtained from the user, listening to any connection from other users (recipients), and applying cryptography protocols; taking into account the agent’s
individual objectives and goals as well as information received from other agents. Agents have the ability to interact, coordinate, and cooperate with each other in order to achieve the overall goal of the system, which is to send secure message to the recipients.

3. The extensible property of the agent allows it to be added or instantiated when a new communication is needed, and deleted when the communication has ended. As a result, the agents can handle multiple communications at once.

4. The mobility characteristic of the agent can be used to carry the message across the network, which allows the agent to carry out tasks on behalf of the user/sender.

From the points discussed so far, we have pointed out the desired characteristics of the agents to support secure communications that are the cooperative and coordinative characteristics to achieve overall system goal, autonomy, extensibility, the interactive ability to support agent-to-agent communication, and mobility. The next section discusses these characteristics in greater detail.

4.3 MAS Suitability for MLC

This section will discuss in details the selected characteristics of the agents discuss in the previous section.

4.3.1 Cooperation and coordination

In MAS, cooperation occurs when two or more agents work together and coordinate their actions in order to achieve the overall system goal. Early work on cooperative distributed problem solving (CDPS) can be seen in the work of Durfee et al (1989). They characterised CDPS as: (a) agents are seen as problem solvers that can work independently (2) an agent cannot complete the overall system’s goal without cooperation with the other agents, (3) cooperation is needed because no agent has sufficient expertise, resources, and information to solve the goal, and (4) different agents might have different expertises to solve different parts of the overall goal.
Doran et al (1997) stated that cooperation occurs when agents’ actions satisfy either one or both of these conditions:

“(1) The agents have a (possibly implicit) goal in common, (which no agent could achieve in isolation) and their actions tend to achieve that goal.

(2) The agents perform actions which enable or achieve not only their own goals, but also the goals of the agents other than themselves”

Agents must cooperate and coordinate their actions because of the following reasons (Durfee, 2001; Jennings, 1996; Lesser, 1999):

- Distributed expertise and capabilities, where every agent has its own specialization or expertise to process data or information. No one agent has sufficient competence, resources, or information to achieve system goals. That is, each agent cannot solve problems by working in isolation. Each agent work together to provide partial solutions, which later can be integrated into a collective solution
- Dependencies between agent’s actions, which require agents to share partial solutions in order to solve overall problem
- There is a need to meet global constraint, which requires every agent to cooperate to provide solutions that satisfy certain condition.

Nwana et al. (1996) classified four coordination techniques such as contracting, multi-agent planning, negotiation, and organizational structure. The following discusses each technique in details.

4.3.1.1 Contracting

A classical coordination technique is the Contract Net Protocol (CNP) (Smith, 1980), which is an auction-inspired protocol in a market-based mechanism. CNP involves making announcement, bids, and award message for resource allocation and tasks sharing. The protocol has negotiation process, where no agent has sufficient information. Therefore the agent must share information to solve problem. The data and controls are geographically distributed, loosely coupled (each agent is concerned more on computational rather than
communication), and asynchronous, where the overloaded agent must find the most appropriate less burdened agent to execute the tasks.

There are two main roles for the agent in the CNP, which are the Manager and Contractor. Manager divides a problem into sub tasks, announces the tasks to finds contractors to execute them, monitors the tasks executions, and later processes the result. Contractor on the other hand executes the tasks, and reports to the Manager when the task is complete. Contractor can also become a Manager to decompose the task into sub tasks and find a contractor to execute them.

For the agents to engage on a contract, Managers announce tasks to the available Contractors. The Contractors evaluate the tasks and start to bid. Managers then evaluate the bids and finally award the best node with the most appropriate bid.

The work of Smith (Smith, 1980) since then have been evolved. Sandholm & Lesser (1995) has extended Smith’s work to consider the self-interest agents, in order to allow the agents to dynamically choose the stage and level of commitment. Sandholm (1993) proposed TRACONET system that enables interaction among agents with different local criteria. The CIA system was modelled based on the Language/Action Perspective, which was implemented in the contracting process (Verharen et al., 1996). They proposed a linguistic concepts and a model for agents communications based on speech act theory. Ferber (1999) proposed time limit or deadline for each task announcement. Varieties of architectures have been proposed. Nwana et al. argued (1996) that even though CNP has been useful to provide dynamic task and resource allocation using bidding, which lead to better agreement, reliable for distributed control, and recovery from failure. However, CNP does not cater for agents with contradictory demands, which does not identify the conflicts or resolve them. They further argued that CNP is communication-intensive and costly.
4.3.1.2 Multi-agent planning

In multi-agent planning, agents form a multi-agent plan that specifies their future actions as well as their interaction in order to achieve their goal. Planning is necessary in order to coordinate their actions and to avoid any conflict or inconsistency of their actions, by knowing what actions and interactions to be taken beforehand. Multi-agent planning can be divided into three categories (Durfee, 2001; Wooldridge, 2002):

1. **Centralized planning for distributed plans**: a centralized planning involves a coordinator agent that decomposes a plan and distributes them to appropriate agents to be executed. When distributing the plan, the coordinator also considers the synchronization of the plan in order to avoid conflicts.

2. **Distributed planning for centralized plans**: a group of agents cooperate to create a centralized plan. The idea is to give each agent with other agent’s plans, so that they can communicate about their individual plan as well as other plans until all conflicts are solved and converge to the global plan.

3. **Distributed planning for distributed plans**: a group of agents cooperate in order to create their own individual plan. Coordination and communication are needed to solve conflicts.

Nwana et al (1996) argued that multi-agent planning is costly, where it needs higher computing and communication resources because the agents share and process a huge amount of information. Moreover, the distributed multi-agent planning is more complex than the centralized planning because there may not be agent that has the overall view of the system.

4.3.1.3 Negotiation

Negotiation and argumentation are the keys for agents to reach agreement (Wooldridge, 2002). A basic definition of negotiation obtained from Bussman & Muller (1992) states that, “negotiation is the communication process of a group of agents in order to reach a mutual accepted agreement on some matter”.

Wooldridge (2002) classifies four components of negotiation that are (1) a negotiation set, which is the environment for which the agent can make a proposal, (2) a protocol that defines
the types of proposals that an agent can make, (3) collections of strategies that the agent will make in a proposal, and (4) a rule, which ensures that, when an agreement is reached, then the negotiations will end up with an agreement deal. Jennings et al. (2001) characterizes negotiation in a simple framework that for a given negotiation, the agents that participate in the negotiation determine the direction of the search. The negotiation can be viewed as a search space allocated to each agent where it is prepared to make an agreement. Each agent must be able to make and responds to a proposal. The agent needs to provide useful feedback on a received proposal. This feedback can be a critique (comments on which parts of the proposal the agent likes or dislikes) or a counter proposal (an alternative proposal generated in response to a proposal). The feedback will guide the agent that makes the proposal to generate a proposal that can lead to an agreement. The types of negotiation methods can be classified into three categories (Nwana et al., 1996):

1. **Plan-based negotiation**: This negotiation is based on multi-agent cooperation to solve conflicts in a group of agents that device a plan. The agents exchange information to execute tasks and resolve conflicts. The negotiation can be centralized or distributed based on the types of multi-agent planning. According to Alder et al. (1989), negotiation and planning are very tightly intertwined because the agents needs information from other agents in order to function effectively and efficiently.

2. **Game theory-based negotiation**: This type of negotiation is based on the game theory to organize the agents. The key elements in this theory are utility function, space of deals, strategies, and negotiation protocols (Bedrouni et al., 2009). The negotiation process is the interaction processes between agents that involve offering or rejecting offer in a deal that can maximise its utility value. Each agent evaluates the offer received from other agent using its own strategies at each negotiation process. Nwana et al. (1996) argued that game theory-based negotiation are not sufficient for real-life industrial application because (1) the agents do not satisfy real world situation because each agent is presumed to be fully rational and have full knowledge of other agents’ preferences in contrast with the real world agents that only have partial knowledge (2) information obtained from a negotiation that involves a large number of agents will become very huge and intractable.

---

34 More on the elements can be found in (Bedrouni et al., 2009)
(3) the agent does not consider past history or any potential implication from agent interactions because it only cares about its current state, (4) all agents have the same internal models and capabilities, (5) the negotiation only occur between two agents, though there are works that address n-agent negotiation (Zlotkin & Rosenschein, 1993).

3. Artificial Intelligence-based negotiation: AI-based negotiation is based on human negotiation strategies, which involve AI techniques such as logic, case-based reasoning, or constraint-directed search (Nwana et al., 1996). For example, Sycara (1991) viewed negotiation as an iterative process. She used case-based reasoning in the iterative process, based on the fact that human negotiation is often based on past history of negotiation experiences. Raeesy et al. (2007) proposed a fuzzy-based model for negotiation, which they claimed can lead to flexible human-like negotiation. Their model consists of two agents pursuing opposite goals, in a way that the more an agent benefit from a negotiation, the more the opponent loses. Each agent is given deadline and scoring function that evaluates each offer.

4.3.1.4 Organizational structure

Organizational structure specifies a set of long-term responsibilities and interaction patterns for the agents (Durfee et al., 1987). The long-term responsibilities describe the agent’s functionality that can guarantee long-term consistency and satisfactory result of the system performance as a whole. While agents perform their responsibilities, there are agents that depend on other agent’s partial solutions to perform their own responsibilities. These types of agents require to be informed of the other agent’s partial solutions so that they can take further actions. An agent does not need any information that does not effect their action. Therefore, apart from specifying the responsibilities, the organizational structure also decides particular agents that are interested or required a partial solution. When any exchanging process of partial solutions of one agent occurs, it will give effect to the other agent’s actions (Durfee, 2001).

Decker (1996) illustrated the Distributed Vehicle Monitoring Testbed (DVMT) using organizational structure to simulate a network of vehicle monitoring nodes, which is the
agents. Each agent has a problem solving capability that senses data through sound to identify, locate, and track patterns of moving vehicles using sensor(s). Each agent has a specific search area to be explored based on its local knowledge and organizational roles. An interest area represents roles and direct communications between agents. An example of partial result sharing can be seen in Durfee et al. (1987)’s work on DVMT, which allowed an agent to share its vehicle map to its neighbouring agents in order to complete the overall task.

Although one organizational structure for an application is not suitable for other applications (Durfee et al., 1987, 1989; Jennings, 1996) the basic approach of organizational structure is to specify the agent’s actions or functionalities, and divide the problem search space among agents, in such a way that particular agents are assigned to specific tasks. Werner (1989) used roles to describe an agent’s state information, permissions, responsibilities, and values of the agent role so that the agent is capable to perform action in its environment. Singh (1990) utilized the concept of strategies to a group of agents, to describe the agents’ intentions or behaviours, which are seen as a set of strategies of its members. The interactions among agents to execute their strategies and their reactive actions determine the agents’ roles in the group. Shoham and Tennenholtz (1992) employed a rather strict concept called social law. A society of agents adopts a set of laws to represent the actions of the agents. To implement this, the programmer that programs the agents, must commit to the law. Wooldridge (2002) described Shoham and Tennenholtz’s work as an offline design where the laws are designed offline and hardwired into agents.

Organizational structure can be adaptive in a dynamically changing environment. The agents can re-evaluate its organizational structure or rearrange it when necessary, to adapt with the changing situation. Ishida et al. (1990) illustrated the dynamically changing environment of a load balancing problem. The agents were able to reorganize themselves when overloaded with tasks by decomposing themselves. This is to allow parallel processing among them, as well as to compose with each other when the tasks are lessen. Horling et al. (2001) used diagnostic system to correct errors or faults by changing the parameters in the organizational structure. Mathieu et al. (2002) exploited the dynamic interactions between agents to adapt to the changing environment.
Nwana et al. (1996) critiqued that the organizational structure has the following weaknesses: (1) organizational structure is useful in a master/slave relationship, where extreme controls can be put on the slave. As a result, such control would decrease other benefits such as speed, reliability, concurrency, robustness, graceful degradation, and minimal bottlenecks; (2) all agents must have the same domain knowledge and thus, this prevent the creation of heterogeneous agent environment, and (3) the designer should ensure careful design of the agents’ roles and actions.

4.3.2 Autonomy and behaviour

The autonomy of an agent is “the agent’s ability to make its own decisions about what activities to do, when to do them, what type of information should be communicated and to whom, and how to assimilate the information received” (Lesser, 1999). In other words, the autonomy characteristic allows agents to do the assigned tasks independently. Each agent has its own behaviour(s).

The agent’s actions or tasks assigned to the agent are performed from within the behaviours of the agent. Communications and coordination between agents, which are performed in order to complete a task, are done and handled through behaviours. When an agent is executed, it automatically performs its behaviour without being invoked by any external entity.

In a modular programming, a function is invoked by a CALL statement, either by another method or another program. In an object-oriented programming, an object is seen as the main entity in a system, which has data structures and functions. The object is considered as a representation of abstraction of passive entities in the real world. An object is considered as passive because the methods in the object are invoked by a message from an external entity (Odell, 2002). Agents, on the other hand, are active. They have control on their actions as they have goals and rules. They know when to act, or update their states compared to traditional approach where it needs to be invoked in order to act or update its state. Autonomous is performed by not providing the agent with call-backs function to its own object reference to other agents, in order to lessen any chance of other entities taking control of its services. As a
result, control complexity is reduced and divided within the agents themselves (Bellifemine et al., 2007; Jennings, 2000).

In the organizational structure where the agent’s actions or functionalities are specified beforehand, the authority and connectivity to the flow of information, and control between agents are determined in terms of topologies (Durfee et al., 1989).

![Figure 4-1: Control Topology](image)

Figure 4-1, an excerpt from (Durfee et al., 1989), describes the control topology of the agent’s organizational structure: (a) flat (lateral) structure, where each agent processes its data and exchanges results among themselves; (b) hierarchical or heterarchical structure, where agent at middle layer provides partial results and sends to the higher layer; and (c) matrix organizations, where different agents have different responsibilities and processing capabilities, and their result are integrated by other agents. According to Nwana et al.(1996), organizational structure is useful when the relationship between agent is in the hierarchical
structure, because more controls can be put over the lower layer agents in order to solve problem. The higher layer agents have more autonomy compared to the lower layer agents that only have partial autonomy. Autonomy can also be correlated with predictability (Huhns & Singh, 1998), that is, an agent is more autonomous if it is less predictable. When more control is put in an agent, the more predictable it appears, and less autonomous it will be.

4.3.3 Extensibility

According to Sycara (1998), MAS is extensible in such a way that “the number and the capabilities of agents working on a problem can be altered”. Extendibility focuses on adding or removing capabilities or skills of an agent to an existing system. Debenham (1999) defines extensibility as “the abilities to easily add new functionality to a system, or upgrading any existing functionality”. To be extensible, MAS should be capable of performing new functionality that it is currently does not able to perform. A new agent representing a new system’s functionality can be added to the system, without reconfiguring the whole system.

Extensibility might sometimes be called scalability, which can be defined as “the ability of a solution to a problem to work when the size of the problem increases” (Rana & Stout, 2000). For example, a SAIRE agent (Odubiyi et al., 1997) is an intelligent searching agent that support public access to Earth and Space Science data over the Internet. Scalability in this system is projected by the ability of the SAIRE agents to be adaptive, to support information access and retrieval from new domains. This implies that the agents can be skilled to adapt new information from different domains. Another example is ShopBot, a comparison-shopping agent that autonomously learns how to shop online (Doorenbos et al., 1997). The scalability portrayed here is the ability of the agent to learn about given online stores, and subsequently adapt to understand different new vendors. However scalability happened when (1) increasing number of agents in a given platform, (2) increasing number of agents across the platforms, (3) increasing the size of load, and (4) increasing the diversity of the agent (Lee et al., 1998; Rana & Stout, 2000). Therefore, systems like SAIRE and ShopBot should be best described as extensible functionalities (Song & Korba, 2002; Wijngaards et al., 2002).
In this research, we focus on the extensible functionalities, where the agent can be specialized or skilled, and added to the existing system. As discussed in the previous section, an agent is autonomous, where they can decide when and how to take actions. Agents can be unpredictable because they can say ‘No’, or reject a proposal given to them. Therefore, agents are independent of each other or loosely coupled (Bellifemine et al., 2007). The advantage of the autonomy and independent traits is that, it is possible to add or remove new agents from the existing system without reconfiguring the whole system. If an agent needs to use the already removed agent’s service, it can use Agent Management System (AMS) and Directory Facilitator (DF) which are the standard services offered by the agent system (Bellifemine et al., 2001; Bellifemine et al., 2007) to search other agents that provide similar service. When adding a new agent, this new agent can simply search other agents within the system and start its interactions. In contrast with the modular programming and OOP, it is difficult to support a loosely coupled feature because a function or an object’s method cannot say ‘No’ to the message, or else, it can produce error to the program.

Many examples of extensibility of the functionality of the agent systems can be found in the literature. Lukose (1997) designed different types of mobile agents such as purely reactive, highly deliberative, or hybrid architecture. Each type of agent is constructed using various components that are integrated and modelled as an executable Problem Solving Method. A mobile agent can be instantiated with the type of agent desired and added to the system.

An agent’s classes or functions can be reused to support extensibility. Pridgen & Julien (2006) developed SMASH, which is an agent platform that provides fixed classes to instantiate secure agent. The security services that should be applied to an agent that arrive at a host include tamper detector that is to check the integrity of the agent, authentication and authorization services. Mosqueira-Rey et al.(2007) designed an agent that analyses packets in the network using a packet sniffer. The agent collects data and creates a data model to be an input to a rule-based inference engine. The packet sniffer class is designed in such a way that it can be instantiated with different specification of a sniffer without reconfiguring the agent.
An agent can be instantiated with different specialization or parameters to represent a new functionality. For example, in data mining, an agent with specific data mining techniques can be instantiated and added to the system temporarily to perform given tasks. For example, a new instance of the agent is created to mine data with classification or association technique (Albashiri et al., 2009). Marks (1999) developed agent-based data mining system that is able to adapt to new data sources without modifying the existing system. An agent can be instantiated with new parameters of the new data sources.

Extensibility can also be seen in simulation system. DrillSim (Massaguer et al., 2006) is a multi-agent simulation system that provides an environment to simulate IT solutions and tests their effectiveness in the context of disaster response. Extensibility is represented by instantiating new agents that represent new simulation scenarios, which are created by assigning a set of roles and physical and cognitive profiles to a large number of agents. In the traffic simulation system proposed by Tao & Huang (2009), mobile agents are instantiated to travel to hosts, to assist in the traffic simulation process.

4.3.4 Interactive

The key element for multi-agent interaction and social organization is communication. Agents are able to cooperate and coordinate their actions to execute tasks through communications. In the organizational structure approach for example, communications are needed to inform agents that are interested or required partial results of the other agents.

4.3.4.1 Agent Communication Language (ACL)

ACL enables an agent to exchange data and information with other agents. According to Genesereth & Ketchpel (1994), there are two different approaches to design the Agent Communication Language (ACL), which are the procedural approach and the declarative approach. The procedural approach is based on the executable code, which can be achieved through programming language such as Java (Arnold & Gosling, 1998) and Telescript (White, 1994). However, this approach has restrictions in term of controlling the information
synchronization and merging. On the other hand, the declarative approach employs words that could be easily understood by human, such as request, query, send, etc.

FIPA-ACL (FIPA-ACL) and KQML (Finin et al., 1997) are the most commonly used agent language. KQML provides message formatting and message-handling standard to support knowledge sharing among agents in a run-time environment (Finin et al., 1994). FIPA-ACL, which is based on the speech act theory, suggest that agent’s actions are represented by messages (Bellifemine et al., 2007).

Both ACLs have similarity in characteristics that are the communication protocol, the format of the message content, and the ontology. Both KQML and FIPA-ACL have a distinct communication protocols known as performative, or communication act. This protocol uses speech act language such as request, send, accept, reject, etc. The format for the message content represents the real content of the information in the message, sent from a sender to a receiver. The value of the content is either in the form of string or a byte sequence (Bellifemine et al., 2007). Meanwhile, the ontology can be defined as “a specification of a conceptualization” (Gruber, 1993). In other words, ontology is a specification to represent shared concepts and their relationship in a domain of interest. It is important for agents to have a common shared ontology so that the message sent can be properly understood by each other. This is essential to successful communication and coordination of the agents.

4.3.4.2 Types of Communications

Two types of communications exist for the agent interactions that are shared data repository and direct message passing.

1. **Shared data repository**: this type of communication can be seen in a blackboard systems (Hayes-Roth, 1985). An agent sends messages such as its partial result to the blackboard, and also gets information from it.

2. **Direct message passing**: the message passing mechanism is widely used in MAS. Agents send messages to one another as a way of communication. Huhns & Stephens (1999) noted that an agent can assume a passive, active, or both roles to function as master, slave, or peer respectively. Each agent must be capable to accept messages. There are
two types of messages that are assertions and queries. A passive agent must be able to answer a question, which is to accept a query or send reply to the requester. An active agent must be able to send a query and make assertions. For a peer agent, the agent must be able to perform active and passive roles. Three most widely used methods of communications are point-to-point, where an agent send a message to another agent; broadcast, where an agent sends a message to all agents in the system; and multicast, where an agent sends a message to a specific group of agents.

4.3.4.3 Message Format
FIPA-ACL and KQML have similar language syntax, message format, as well as message parameters. However, FIPA-ACL is restricted to FIPA architecture and has additional functionalities, such as agent management. FIPA-ACL treats commands like register, unregister, recommend, broker, etc as reserved words and therefore agents are able to request for these commands. As a result, the FIPA-ACL message does not have to use the commands such as ‘register’ (see Figure 4-2, from (Brien & Nicol, 1998)) unlike the KQML message (see Figure 4-3, from (Flores-Mendez, 1999)).

```
(INFORM
  :sender bt-agent
  :receiver customer-agent
  :content
    (Line_quote(bt_customer,123),300)
  :in-reply-to round-4
  :language prolog
  :ontology bt-auction
  :protocol fipa-contract-net
)
```

Figure 4-2: Example of FIPA-ACL message
4.3.5 Mobile

One of the distinguished characteristics of the agent is mobility, which is the ability of the agent to migrate or move from home platform to another platform, carrying its code and data.

4.3.5.1 The mobile agent paradigm and distributed computing

Distributed computing can traditionally be realized in three ways that are Client-server paradigm, Remote-evaluation paradigm, and Code-on-demand paradigm (Bellifemine et al., 2007; Braun & Rossak, 2004; Carzaniga et al., 1997). Client-server is the most widely used paradigm. Generally, one or more clients request service(s) to a server. This is because the client does not have the resources or the knowledge to produce the service. The server responds by processing the request and delivering the service to the client. In the Remote-evaluation paradigm, the client has the knowledge about service, but it does not have any resource to produce it. Therefore, the client sends a request in a form of code, so that the code can be executed at the server. Then the server sends the result back to the requested client. The Code-on-demand paradigm describes a situation where the client has the resources, but it does not have the knowledge on how to produce the service. Thus, the client sends a request for a code from the server, and executes it on its side.

The mobile agent paradigm on the other hand, describes a client that has the knowledge about the required service, but only has partial resources for it to complete the processes to produce the service. Therefore the client interacts with a server to transmit its code. The server
executes the code and provides the required resources. (Braun & Rossak, 2004; Huhns & Singh, 1998) characterised mobile agents like the following:

- Mobile agents are used in the wide-area and heterogeneous networks where there is no reliability on the connection or the security of the network.
- The migration of the agents is initiated by the agent (or programmer). This is contrary to the mobile object systems where the migration of the object is initiated by the operating systems or middleware.
- The agents migrate to access resources only available at the remote hosts.
- Mobile agents are capable of multi-hopping, or migrating to more than one hosts, contrary to remote-evaluation and code-on-demand paradigms that can only be migrated once.

4.3.5.2 Structure of mobile agents

A mobile agent consists of three fundamental elements that are code, state, and data (Bellifemine et al., 2007; Braun & Rossak, 2004). When the agent migrates, the code is executed at the remote platform. The state preserves the environment and data that the agent needs for the execution. The data describes the parameters or variables that are brought along with the agent to be used in the execution process. When migrating, the agent invokes a method for migration. For example, in Jade (Bellifemine et al., 2007), a method called doMove() is used, which permits the agent to migrate from its home platform to a remote platform. Once migrated, the code will be executed in the remote platform and the original agent is destroyed. In Tracy Mobile Agent (Braun & Rossak, 2004), the go() method is used to migrate the agent. By calling this method, all of the agent’s executions are stopped and the statement after the go() method will never be executed. The remote platform is responsible to recreate the agent and allow it to execute once it arrives there. Mobile agents are provided with the ability to communicate with agents in the remote platform to negotiate and ask for resources. There are two types of mobility that are strong and weak mobility (Bellifemine et al., 2007; Braun & Rossak, 2004). Strong mobility includes the agent’s current state when migrating. At the recipient’s platform, the agent will continue to execute from the very next
instruction of the agent’s code. Weak migration does not include the agent’s state. At the recipient’s platform, the agent’s execution starts from the beginning of the code.

4.3.5.3 Advantages of mobile agents
The advantages of mobile agents are like the following (Braun & Rossak, 2004; Huhns & Singh, 1998; Jain et al., 2000):

- Mobile agents can be used for dynamic code installations, for example in software update.
- Mobile agents are suitable to work in the offline mode, for example in the low processing power devices such as PDA, the agent that work on a PDA can migrate to another host that is online if the owner’s host is turn off.
- Sending a mobile agent to be executed at remote servers can benefit users, especially for processes that involve large amount of data that cannot be transferred back to the mobile agent’s original platform.
- Mobile agents can help improve code modularity and reusability, and they can help hide network, system, and protocol heterogeneity.
- A mobile agent can act on behalf of its user without permanent contact or control from the user. Thus, the user can save time to do other important tasks.

4.3.5.4 The security concern and solution approach to mobile agent security
The mobile agent approach has been successfully supporting various applications such as in the security system (Aslam et al., 2001; Nagesh, 2006), electronic commerce (Sandholm & Huai, 2000), or in the information retrieval (Qu et al., 2008). However, mobile agents paradigm arises security issues, especially on how to protect the mobile agent from other agents in the platform, or from the platform it arrives at (Bierman & Cloete, 2002; Jansen & Karygiannis, 1999; Sander & Tschudin, 1998). An agent that arrives at a platform is vulnerable to other agents in the platform. A malicious agent can launch an attack to another agent in the form of masquerading, denial of service, repudiation, and unauthorized access. A malicious agent is capable of masquerading or pretending to assume a fake role and convince a legal agent to involve in a negotiation; or continue sending message to a legal agent as a
denial of service, so that the legal agent cannot perform its actions; or refuse to acknowledge a certain legal action has took place; or interfering with and modifying the legal agent’s actions, data, or code, if the platform has weak or no control mechanism.

A mobile agent is also vulnerable to threats posed by the platform it arrives at, such as masquerading, denial of service, eavesdropping, and alteration. A malicious platform can fake its identity to be a legal one in an attempt to harm the agent; or deny or delay any service request, or refuse to execute a code, or terminate any agent that arrive in the platform; or eavesdrop the agent activities, code, or data; or because the platform has access to the agent's resources, it can simply alter the agent code and data.

Because of these threats, it is important to prevent or detect malicious modification or alteration to the agent’s code and data. The security requirements for mobile agents are generally similar such as discussed in Subsection 3.3.2.1. Authentication, integrity, confidentiality, non-repudiation, and anonymity are required to secure mobile agents (Huhns & Singh, 1998; Jansen & Karygiannis, 1999). However, the security of a mobile agent against malicious platform is difficult to achieve because, to allow a mobile agent’s code to be executed is to allow both the code and data to be exposed to the platform (Farmer et al., 1996; Huhns & Singh, 1998). Due to these threats, mechanisms to protect mobile agents have been developed and researched. Generally, the mechanisms can guarantee the following (Farmer et al., 1996): (1) allow a platform to authenticate a mobile agent that arrive, (2) allow a platform to verify the integrity of the agent, (3) ensure confidentiality of the agent during transmission, and (4), allow/disallow the agent from accessing resources at the platform. Many cryptographic techniques have been developed to counter threats to mobile agents, which stress on the prevention or detection of threats.

Examples of studies that emphasize on preventing threats to mobile agent include Sliding encryption (Young & Yung, 1997), which is used to provide an economic solution by encrypting small chunks of data into small chunks of ciphertext using a public key. The data is encrypted with a public key, which the agent carries to the targeted platforms. After finishing its itinerary, the agent migrates back to the agent’s original platform. The decryption processes
only occur at the agent's original platform where the private key resides, to prevent any unauthorized party to recover the plaintext.

Another technique to prevent threats are presented in (Sander & Tschudin, 1998) that proposed a secret way for a mobile agent to execute a code, in such a way that the other agent or the agent platform cannot learn the code. The sender encrypts a function that is to remain secret to be $E(f)$. Then, a program that is designed to implement $E(f)$, $P(E(f))$ will be embedded within the agent. The agent is executed at the recipient’s platform to run the program on $x$: $P(E(f))(x)$, and the result is returned to the sender. This way, the main purpose of the agent (that is to calculate $E(f)(x)$) is hidden away from the recipient.

There is also a technique that allow mobile agents to construct cryptographic key based on the observed environmental data (Riordan & Schneier, 1998). The agent carries encrypted data that can only be decrypted if some environmental conditions are true. The agent is able to search specific information to trigger the key generation. The environmental data is kept in the form of hash message or a steganographically hidden message, to prevent malicious hosts or malicious agents to directly learn the agent's code and actions. However, if a malicious host has a complete control over the environment, it can simply modify the agent’s code to execute other actions that are contrary to the legal actions of the agent, or deny any environmental resources to the agent. In addition, a host might limit the code execution because of possible unsafe operations (Jansen & Karygiannis, 1999).

A unique prevention technique is presented in (Ametller et al., 2004a, 2004b; Pedro Manuel et al., 2006)\(^{35}\), which design a mobile agent that can protect itself. To prevent malicious code injection to the data carried by the agent, it carries its own protection by using disposable key pair (public and private keys). The sender’s agent signs the data using the private key and produces a signature. The corresponding public key is embedded into the agent’s code. The signature and a hash of the agent code are encrypted with the recipient’s public key and sent to the recipient. At the recipient’s host, the mobile agent code is authenticated using the hash.

\(^{35}\) Refer to Figure 2-12
message. If the code is valid, it is executed to make a request for a service. The integrity of the agent’s data can be checked by verifying the signature with the public key.

There are also mechanisms that provide a sender a method to detect modifications on the mobile code executed at the remote hosts. A tracing mechanism is presented in (Vigna, 1997), which consist a log tracing in the form of a unique identifier of a statement and a signature. When the targeted platform has finished executing a code, it produced a tracing log. Then, the result of the tracing is sent back to the owner of the agent so that the owner can make sure that the code has been executed correctly. Any modification to the code can be detected. However, the log tracing size that must be stored and carried by mobile agent will grow in time and should be limited.

Grimley & Monroe (1999b) proposed the ‘phone home’ approach that describes a method for a mobile agent to contact a sender, and tell the sender the current state it was in, or to transfer important data home. Therefore, any tampering with the agent can be detected by the sender by checking the data received from the agent. Various other techniques have been researched to secure mobile agents. Jansen & Karygiannis (1999) present an extensive studies on the threats to mobile agents as well as the countermeasures.

4.4 Justification for MAS

We have so far discussed the advantages of the MAS’s characteristics to support the traditional non-agent based approach. In this section, we justify why the MAS-based system approach is suitable to cater for the security processes proposed in MLC. We call our MAS-based system as MAgSeM (discussed in Chapter 5, Section 5.3.4). We are interested in the MAS characteristics such as the ability to coordinate and cooperate to achieve the overall system goal, autonomous, extensible, interactive, and mobile.

MAS complies with FIPA standard, which provides agent development standards and specifications, including agent communications and agent management (Bellifemine et al., 2007). This standard provides a systematic and convenient way to develop agent
communications with the readily available message format, performatives, as well as the protocols to send messages. Messages to and from agents can easily be handled using a standard message templates. Besides, the standard allows agents to be implemented in the same way, even on different devices.

In contrast with the traditional approach, programmers need to start developing communication or message exchange programs from scratch. This includes handling message passing as well as protocols to send message (such as socket programming (Harold, 1997)). In addition, without any standards available, the implementation of the communication and message exchange programs are different with different devices.

As MAgSeM utilizes different types of devices such as PDAs, smart phones, in addition to PCs, the MAS approach is deemed suited for the development of MAgSeM, where the codes that are implemented for PC can be reused with minimal adaptation for mobile devices.

In MAS, the agents are autonomous and can take initiative by having its own state and data. By default, once an agent is executed, it will perform tasks such as specified in its behaviour, without being invoked by other agents or any external objects. An object on the other hand, once it is instantiated from a class, it needs an external entity to pass a message to its method in order to invoke it. As a consequence, the object-oriented approach is not suitable to be used in systems that focus on automating its processes. The MAS approach is more suitable because agents can work with minimal intervention from the user. For example, an agent could perform the whole task either by itself or by cooperating with the other agents without being invoked by any external entity. It could also instantiate a new agent with different skills to perform tasks in order to achieve the overall system’s goal. Moreover, an agent can take initiative to migrate to another host. This is contrary to a mobile object system where the migration of the object is initiated by the operating systems or middleware (Braun & Rossak, 2004).

The autonomous characteristic of the agent gives an advantage to automate and cater for the security processes. The sender’s agent collaborates with the agent from the recipient’s side in
order to get permission to send a message and to determine what kind of algorithm that should be used in the communication session. In addition, a mobile agent can be launched to carry a secure message to the recipient’s side and let the agent handles the security processes there, without having the user to interfere with its processes.

In summary, the MAS approach is used in order to provide the MAgSeM-based system, with flexible security components, which offers different types of security levels for different types of communications. The agents cooperate and coordinate with each other to achieve the overall goal of the system, which include organizing information obtained from the user, listening to any connections from other users (recipients), and applying appropriate cryptography protocols. The extensible property of the agent allows it to be added or instantiated when a new skill is needed, and deleted when it has finished its tasks. As a result, the agents can handle multiple security processes at once.

4.5 Summary

The MAS approach offers added value on top of the non-agent or the traditional system approach through its agents’ characteristics such as the ability to coordinate and cooperate to achieve the overall system goal, autonomous, extensible, interactive, and mobile. We have discussed the suitability of these characteristics to support security in distributed systems with our MLC model. These characteristic have been proven useful to support traditional approach through the examples discussed above. In the next chapter, we discuss our proposed model based on the MAS approach and the MLC model that utilizes the discussed characteristics.
CHAPTER 5 PROPOSED MULTI-AGENT SECURITY MODEL

5.1 Introduction

This chapter presents the proposed model, which is based on MAS, called the Multi-agent Security Model (MAgSeM). The goal for MAgSeM is to provide a secure environment for online communication between a sender and a recipient motivated by the needs for secure communications in e-health. MAgSeM caters for the distributed security processes in the communications using MAS characteristics, such as autonomous, extensible, interactive, mobile, and the ability to coordinate and cooperate to achieve the overall system goal. By using the MAS approach, MAgSeM provides a security model that offers control on the sender’s side using the mobility and extensibility characteristics of the agent.

MAgSeM applies the MLC model (detail in Section 3.2.2) as its security mechanism. We first identify the actions or tasks that each agent has, using the Organizational Structure coordination techniques\(^\text{36}\). Then, we describe each agent’s action, and how the agent uses the MLC specification in order to secure the information from a sender to a recipient. We also explain how our mobile agent transfers the information in a secure way, while maintaining control over of the information.

\(^{36}\) Refer to Section 4.3.1.4
In Section 5.2 the Chapter starts by defining the agent’s goals and sub goals using the Organizational Structure technique. Then, in Section 5.3, we present the proposed architecture of MAgSeM and the implementation of the MLC model. We explain how we provide control of the data at the sender’s side using the MAS approach. Afterwards, in Section 5.4, we describe the communication architecture in MAgSeM in further detail, where each agent’s actions are described in detail. Finally, we summarize this chapter in Section 5.5.

5.2 Identifying Agents Goals against Organizational Structure

We use the Organizational Structure for the agents’ coordination technique, where we specify each agent’s actions or functionalities that are assigned to them for completing an overall goal. According to Jennings (1996), in this technique, the actions of agents in solving goals can be expressed through a classical AND/OR graph (Mahanti & Bagchi, 1985).

We identify the overall multi-agent system goal and the sub goals using the AND/OR graph. These goals represent the agent’s actions. Figure 5-1 shows the AND/OR graph to illustrate the goal and sub goals. In MAgSeM, there are two main goals that are G, which is “Send Secured Data to Recipient” and G’, which is “Listen to Incoming Requests”.

We divide G into G1 to G4, and assign an agent into each sub goal, labelled as A1 to A4. G’ is divided into G’1 and we assign A7 to G’1. These sub goals may or may not contain another level of sub-goals. Agents that are drawn with double lines indicate agents that are instantiated to complete certain sub goals. For example, A5 and A6 are instantiated by A4 in order to complete G4.

An ‘AND’ node can only be completed, if all of the immediate sub-goals are completed. For example, G4.2 (Apply Cryptography Protocols) can only be completed if goals G4.2.1 to G4.2.4 are completed first\(^37\). An ‘OR’ node can be completed by choosing either one of the sub-goals. For example, A8 may entertain a request by performing either G’1.1.1 (Verify

\(^{37}\) Previously in Section 3.3.2.3, we have described the step-by-step processes of Mechanism 1: Data Security (at the sender side). These steps are represented by G8 until G12.
plaintext), or G’1.1.2 (Sign token), or G’1.1.3 (Process received data), provided that all conditions are met.

Figure 5-1: AND/OR graph for the agent's actions
The graph illustrates the interdependencies between goals and data/resources, which are needed to solve the primitive goals. The solid arrows indicate interdependencies between goals and data/resources, which is drawn in bold lines.

The graph also illustrates interdependencies between goals. The dotted arrows indicate interdependencies between G4.2 and G4.3, which are *Apply Cryptography Protocols* and *Send Data*. In an organizational structure, it is important for each agent to report each result of the sub-goals, or report that it has finished its goal to the other agents that need the result, so that other agents could use the results to determine their next actions. For example, A5 must report its end result to A6, so then only A6 can send the secured data to the recipient’s side.

5.2.1 Organizing the Agents

![Figure 5-2: Organizing the agents in the layered architecture](image-url)
From Figure 5-1, the agents can be summarised and organized as a layered structure (as depicted in Figure 5-2). The architecture classifies agents that execute the following functionalities:

1. Interact with the user: authenticate a user to enter the system and acquire data from the user
2. Organize information and send request: keep addresses of other users, request certificates, and keep record of undelivered message
3. Apply cryptography protocols such as described in Section 3.3.2.2
4. Dispatch mobile agent to the recipient’s platform
5. Listen to requests from other users: respond to a request from other users’ agents and verify the authentication of the information received from the agents.

From Figure 5-2, the agents are renamed with meaningful names as follows:

A1:\textbf{ Interface Agent} (IA): interacts and obtains data from the user
A2: \textbf{Data Organizer Agent} (DOA): organizes the data received from the user such as the message and the recipient’s address
A3: \textbf{Multi-tasking Agent} (MTA): makes a request to send a message to other users and keeps track of undelivered messages
A4: \textbf{Crypto Agent} (cA): provides all necessary information and parameters for the security processes
A5: \textbf{SetUp Agent} (SUA): applies cryptography protocols
A6: \textbf{Mobile Agent} (MA): carries secured data to the Recipient’s platform
A7: \textbf{Communication Listener Agent} (CLA): listens to any incoming request
A8: \textbf{Receiver Agent} (RA): provides verification service to the agents that arrive at the platform

The agents cooperate with each other to perform security processes, by sending reports and messages (or partial results) to the other agents in order to achieve the overall goal. Agents at the lower layer send their partial results or reports to the agents at the higher layer. A report
indicates that an agent has finished its task. The partial results are integrated at every layer in the processes of generating an overall goal.

5.3 MAgSeM Architecture

We proposed a multi-agent architecture called MAgSeM to cater for the security processes at the sender’s and recipient’s side. The agents in the previous Section 5.2.2 are integrated in the model, and another two agents are added, which are (1) **Server Agent** (SvA): resides at the server’s side to manage the authentication process and other requests from agents; and (2) **Decrypt Agent** (DA): performs the decryption process at the recipient’s side.

---

**Figure 5-3: Proposed MAgSeM**

---
Figure 5-3 shows the proposed MAgSeM. At the sender’s side, IA sends the ID, password, and IP address of the sender to SvA to be authenticated\(^{38}\). SvA authenticates the user, and if the sender is authorized, it sends the authentication result (valid/invalid) as well as a list of IP addresses of other users that have exchanged certificates with the sender.

Then, IA gets the data and address of the recipient(s) from the sender and gives it to DOA. DOA organizes the data into a file or plaintext\(^{39}\) and gives both plaintext and the address to MTA. The *Secure Communication Layer* is where the security processes are performed. At this layer, MTA sends a request to send a message to the intended recipient(s). cA prepares all necessary information for the security processes and determines the appropriate layer for the communication as described in the MLC specification. SUA is instantiated by cA to apply the cryptography protocols. MA is instantiated by SUA to send the data to the recipient by migrating to the recipient’s host.

At the recipient’s side, CLA listens for any incoming request to send a message from MTA. If there is a request, CLA instantiates RA to entertain the incoming MA. MA instantiates DA at the recipient’s side to perform decryption processes. At the server side, SvA compares the ID and password with the one in the database. If they are matched, SvA stores the IP address of the user.

MAgSeM supports the implementation of MLC such as described in Section 3.3.2, and provides a rather generic architecture that can be used in any type of domains. The agents use the autonomous, extensible, interactive, and mobile characteristics and are able to coordinate and cooperate with each other to achieve the overall system goal.

Each agent is skilled to perform certain specific tasks. An agent can be extended or instantiated by using the extensibility characteristic, when its skill is required for the system to

\(^{38}\) Assumption is made that the certificates of all users have been exchanged beforehand. A security administrator in an organization could be responsible for managing certification exchanges.

\(^{39}\) The data from user will hereafter be called plaintext
achieve the overall goal. For example, SUA is instantiated for every recipient’s communication to apply the desired cryptography protocols. DA is instantiated only when the condition is met to decrypt an encrypted message.

Agents in MAgSeM cater for and automate the security processes with minimal intervention from the user. MAgSeM uses the mobile agent approach. It is used to carry secured data to the recipient. A mobile agent is suitable for a situation where the agent has only partial resources at the home platform, and the rest of the resources are located at the recipient side. In our case, the mobile agent requires the private key of the recipient, which is only available at the recipient’s host to decrypt the secured data. A mobile agent is robust, in a sense that if the destination platform is shut down while the agent is still there, the agent can take necessary actions such as migrating back or terminating its activities (Lange & Mitsuru, 1999). It can send a notice to the home platform about its situation and terminate if required.

5.3.1 Communication Layers

For a communication between two users say, Doctor and Nurse, the layer of communication or \texttt{com\_layer} is identified. \texttt{com\_layer} refers to the five layers of communications in the MLC model, which determines the security mechanisms and symmetric key lengths that will be applied to information in a particular communication session. \texttt{com\_layer} can be determined by using a default layer value or \texttt{L0}, assigned to each user, like the following:

- Patient, Doctor, and Nurse : \texttt{L0} = Layer 1
- Paramedic and System Coordinator : \texttt{L0} = Layer 2
- Social Worker : \texttt{L0} = Layer 3
- System Administrator : \texttt{L0} = Layer 4.

The assignment of \texttt{L0} is based on the sensitivity of the data each user may carry. Smaller \texttt{L0} is assigned to users that communicate extremely sensitive information, while a bigger value of \texttt{L0} is assigned to users that communicate low sensitive information. \texttt{L0} is obtained using a
user’s certificate. Each certificate in the server is named with the users’ name and the default layer. Therefore, a default layer for a user can be obtained by extracting the value from the certificate.

The following describes the rules to determine com_layer for a communication between a sender and a recipient:

1. If L₀ for the sender and recipient are the same, then com_layer for that communication will be the recipient’s L₀.
2. If L₀ for the sender is greater than the recipient’s, then com_layer for that communication is the sender’s L₀.
3. If L₀ for the sender is smaller than the recipient’s, then com_layer for that communication is the recipient’s L₀.

In summary, com_layer can be identified by comparing both L₀s of the sender and the recipient. The one with a larger value will be chosen as com_layer. For example, in SW (L₀ = 3) ⇔ Doctor (L₀ = 1) communication, the com_layer will be Layer 3. If both L₀s are the same, that L₀ will be used as the com_layer value. After com_layer is identified, the security mechanisms for the communication can be determined, that is, whether the communication needs data security, or channel security, or both data and channel security. The com_layer is associated with the length of the symmetric key encryption algorithms such as proposed in Section 3.3.2.3:

- **Layer 1**: key length = 193-bit key and longer
- **Layer 2**: key length = 129-bit to 192-bit key (wireless: 80-bit to 192-bit key)
- **Layer 3**: key length = 112-bit to 128-bit key
- **Layer 4**: key length = 80 to less than 111-bit of key

---

40 Refer to Section 6.3.3.1
5.3.2 MLC Specification

The MLC Specification stores the security specifications, which describes the information about the symmetric key encryption. MLC specification is stored as a tuple containing four parameters:

\[ <\text{Algorithm}, \text{lengths}, \text{mode}, \text{padding}> \]

Algorithm, lengths, mode, and padding describe the types of algorithms for the symmetric key, the lengths of the key, encryption modes, and encryption padding respectively.

<table>
<thead>
<tr>
<th>Alg</th>
<th>Length</th>
<th>Mode</th>
<th>Padding</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-256-CBC-7</td>
<td>256-bit</td>
<td>CBC</td>
<td>PKCS7</td>
</tr>
<tr>
<td>AES-128-CBC-7</td>
<td>128-bit</td>
<td>CBC</td>
<td>PKCS7</td>
</tr>
<tr>
<td>3DES-168-CBC-7</td>
<td>168-bit</td>
<td>CBC</td>
<td>PKCS7</td>
</tr>
<tr>
<td>AES-192-CBC-7</td>
<td>192-bit</td>
<td>CBC</td>
<td>PKCS7</td>
</tr>
<tr>
<td>BLO-112-CBC-7</td>
<td>112-bit</td>
<td>CBC</td>
<td>PKCS7</td>
</tr>
<tr>
<td>BLO-80-CBC-7</td>
<td>80-bit</td>
<td>CBC</td>
<td>PKCS7</td>
</tr>
<tr>
<td>TWO-128-CBC-7</td>
<td>128-bit</td>
<td>CBC</td>
<td>PKCS7</td>
</tr>
<tr>
<td>TEA-128-CBC-7</td>
<td>128-bit</td>
<td>CBC</td>
<td>PKCS7</td>
</tr>
</tbody>
</table>

Figure 5-4: An example of MLC specifications

Figure 5-4 gives an example of a set of MLC specifications. Examples of algorithms that can currently be used are AES (FIPS197, 2001) 256-bit and 192-bit representing Layer 1, Triple-DES 168-bit (Barker, 2008) for Layer 2, AES 128-bit, Twofish 128-bit (Schneier et al., 1998), TEA 128-bit (Wheeler & Needham, 1994), and Blowfish 112-bit (Schneier, 1994) represents Layer 3, and Blowfish 80-bit represents Layer 4. The encryption mode is cipher-block chaining mode (CBC) and padded with PKCS7 padding. The encryption algorithm is selected randomly by the agent, after com_layer has been calculated as part of the communication process.

Any security algorithm can be added to MAgSeM, as long as it complies with the security specification in the MLC Model described in Section 3.3.2.3. The method of selecting the algorithms presented here is more flexible compared to the method in the existing technologies discussed in Section 2.6. MAgSeM attempts to provide different security levels for different types of communication through MLC, which is the opposite of the static implementation of security mechanisms in the existing technologies.
5.3.3 Control over Data by Sender

MAgSeM focuses on a control mechanism on how a sender can securely transfer data to a recipient while maintaining control over the data. The ‘maintaining control’ over the data can be described as:

1. If the message carried by the sender’s mobile agent is seized by an attacker, the attacker still cannot recover the plaintext
2. The recipient or any other third party does not need to know the details of the decryption processes to recover the plaintext.

One way for the sender to gain control over the data, is to keep part of the requirements for the decryption processes a secret, such as part of the agent’s code, or parameters used for decryption. As discussed in Section 3.3.2.2, a symmetric key, $K$ is used to encrypt the plaintext. This key and the information about the key (which is stored in the MLC specification), are kept with the sender until he/she knows that the mobile agent which has moved to the recipient’s host needs it.

A token, which is an encrypted random number, is carried by the mobile agent to the recipient’s host. It is used as a ‘phone home’ mechanism\(^{41}\) (Grimley & Monroe, 1999a), where the agent sends the token back to the sender. This is a way for the agent to tell the sender that it wants the information kept at the sender’s side for the decryption processes.

MAgSeM implements the control mechanism using the mobility and the extensibility of the agents. We revisit Section 3.3.2.2 in the MLC’s Mechanism 1 (Data Security), and modify the basic steps of the cryptography protocols, in order to tailor and adapt with MAgSeM. Two symmetric keys ($K1$ and $K2$) are used (instead of one key) in the security processes. Figure 5-5 describes the step by step processes for the control mechanism.

---

\(^{41}\) According to Grimley & Monroe (1999), phone home mechanism can be used by the mobile agent to contact the sender, either to transfer data to the sender, or simply used as a method to report to sender of the state/wellbeing of the agent.
A sender is represented by Sender Agent. It encrypts plaintext with $K1$ to get a ciphertext. An agent’s code, which has the ability to decrypt the ciphertext, is encrypted with $K2$. A token $T$ is generated to be carried by a mobile agent (MA). MA is dispatched while carrying secure data, which contains ciphertext, encrypted agent’s code, $K2$, and $T$. $K1$ is kept secret at the sender’s side.

(2) At the recipient’s side, $K2$ is shared with the Recipient Agent, so that the agent can use it to decrypt the code, and later use the code to decrypt the ciphertext. However, the key for decryption ($K1$) and the information about the key (key length, mode, and padding) is kept secret at the sender’s side. This is where the token is sent back to the Sender Agent to get the necessary information.

(3) Once $T$ is received, the Sender Agent sends the secret information regarding $K1$ to MA, and then the decryption process to retrieve plaintext is performed.

$K1$ and $K2$ can be fixed to any type of algorithms with the key lengths according to MLC. These steps will be described in details in the next section.
5.3.4 Security Mechanism in MAgSeM

We use the communication between Sender Agent and Recipient Agent, which uses a mobile agent to carry the message from the sender’s side to the recipient’s side (Figure 5-5). Then, we modify the Mechanism 1 (Data Security) in Section 3.3.2.2, to adapt with the mobile agent paradigm and to accomplish our control mechanism in MAgSeM. An agent’s code, which has the functionality to decrypt a ciphertext, is labelled as $Cd$. The following symbols will be used throughout this chapter to explain the security processes:

a. Public and Private keys of the recipient: $(pubKr, privKr)$
b. Public and Private keys of the sender: $(pubKs, privKs)$
c. Symmetric keys: $K1, K2$
d. Disposable secret and public key: $(Ks, Kp)$
e. Plaintext: $P$
f. Hash of $P$: $H(P)$
g. Ciphertext: $C$
h. Signature: $S$
i. Agent’s code: $Cd$
j. Hash of $Cd$: $H(Cd)$
k. A random number $Rand$
l. Token: $T$
m. The information extracted from the MLC Specification, $mlc$

5.3.4.1 Steps taken by Sender Agent

1. Takes the recipient’s certificate and extracts $pubKr$.
2. Calculates com_layer for the sender and recipient’s communication to obtain $mlc$
3. Generates two symmetric keys ($K1, K2$) according to $mlc$
4. Encrypts $P$ with $K1$ to produce a ciphertext
   \[ C = E(P)K1 \]
5. Generates $Rand$, and encrypts it with $K1$ to produce $T$ that will be carried by MA. Sender Agent will keep $K1$ until $T$ is received.
   \[ T = E(Rand)K1 \]
6. Generates disposable secret and public key \((K_s, K_p)\). After \(T\) is received, \(K_s\) is used to encrypt the information that is kept for decryption processes. The corresponding \(K_p\) will be embedded in \(C_d\) and sent to the recipient’s host to be used for decryption\(^{42}\).

7. Take \(C_d\) and create a .jar\(^{43}\) file

8. Signs the .jar file \((C_d)\) with \(privK_s\) to produce a signature, \(S\), which is used to verify that \(C_d\) is from the sender.

\[ S = E(C_d)_{privK_s} \]

9. Encrypts \(C_d, S\), and \(T\) with \(K_2\) to produce Ciphercode.

\[ Ciphercode = E(C_d, S, T)_{K_2} \]

10. To allow only Receiver Agent to retrieve \(K_2\), it is encrypted with \(pubK_r\) together with \(H(C_d)\) to produce Cipherkey.

\[ Cipherkey = E(K_2, H(C_d))_{pubK_r} \]

At the recipient’s side, a new \(H(C_d)\) can later be computed from \(C_d\) in Step (9), and compare it with the one in Cipherkey to check whether \(C_d\) is valid and not violated.


12. Having finished preparing the message, Sender Agent creates an instance of MA to carry the message to the recipient’s host.

13. Waits for \(T\) from MA. Once it is received, produce hashKey, which is the information to be given to MA that contains \(H(P), K_1,\) and \(mlc\).

\[ hashKey = E(K_1, mlc, H(P))_{K_s} \]

\(^{42}\) The generation of \((K_p, K_s)\) is one time per communication session. These keys will be disposed once the communication session is over. This is to avoid any third party from using \(K_p\) (which can be retrieved from the Recipient’s host) in the next communication sessions.

\(^{43}\) Java archive file (http://java.sun.com/javase/6/docs/technotes/tools/windows/jar.html)

\(^{44}\) The signature needs to be encrypted because an impostor, who is also trusted by the recipient can remove \(S\), add his/her own signature, take the agent’s data, and add it to his/her own agent. Then this agent will be sent to the recipient without the sender or recipient knowing that the agent is actually comes from the impostor. This is described as an attack on a targeted state (Roth, 2002).
5.3.4.2 Steps taken by Mobile Agent

The mobile agent carries the message to the recipient’s host, and there it communicates with Recipient Agent. The following describes the communications between MA and Recipient Agent:

1. MA makes a request to process the message
2. Receives result from Recipient Agent indicating that both $Cd$ and $S$ are ‘Valid’/‘Invalid’. If ‘Invalid’ message is received, report to Sender Agent and terminates
3. If both are valid, MA makes a request to sign $T$
4. Sends the signed $T$ back to Sender Agent
5. Receives hashKey from Sender Agent and un-jarred $Cd$
6. Makes a request to execute $Cd$
7. hashKey and ciphertext are passed to $Cd$ for the decryption processes

5.3.4.3 Steps taken by Recipient Agent

Recipient Agent will be in charge of communicating with MA and $Cd$ in the process of decrypting a message.

1. Waits for any request to process messages from MA.
2. Once received, the message is split into Ciphertext, Ciphercode, and Cipherkey
3. Gets privKr to decrypt Cipherkey and obtain $K2$ and $H(Cd)$
   \[ D(\text{Cipherkey}) \text{privKr} = K2, H(Cd) \]
4. Use $K2$ to decrypt Ciphercode to obtain $T$, $S$ and $Cd$
   \[ D(\text{Ciphercode})K2 = T, S, Cd \]
5. Both $S$ and $Cd$ will be verified:
   a. Validate $S$ against $Cd$ using the sender’s pubKs
   b. Recalculate $H(Cd)$ from $Cd$ in 4, and compare it with $H(Cd)$ in 3.
6. If both $S$ and $H(Cd)$ are valid, sends a report to MA
   If one or both are invalid, send a report to MA and abort current process.
7. Sign $T$, when a request is made from MA.
8. When the plaintext $P$, and $H(P)$ are received, recalculate $H(P)$ and check if $P$ is tampered.
9. Sends and reports to $Cd$ whether $P$ is ‘Valid’/’Invalid’
10. If $P$ is valid, notify the recipient.

5.3.4.4 Steps taken by $Cd$

1. Once executed, $Cd$ decrypts $hashKey$ using $Kp$ to obtain $H(P)$, symmetric key $K1$, and $mlc$.
   
   $$D(hashKey)Kp = H(P), K1, mlc$$
2. Loads and recreates $K1$ with $mlc$ to decrypt the ciphertext, $C$
3. Decrypts $C$ to get $P$.
   
   $$D(C)K1 = P$$
4. Sends a report to RA about $P$ and $H(P)$, so that RA can recalculate the hash and validate the plaintext.
5. Terminates itself.

5.3.5 Advantages of the Control Mechanism

By keeping $K1$ a secret, the sender has the advantages of gaining control over the data carried by MA. This is because a recipient or any third party does not need to know about the details of the encryption key. Even though an attacker could intercept $hashKey$ that came from Sender Agent, the attacker still cannot recover the plaintext as the key to decrypt $hashKey$, which is $Kp$, is at the recipient’s host. In addition, $Kp$ is disposable, which is created only once per every communication. $Kp$ and the corresponding $Ks$ will be removed once a communication session is terminated.

Another advantage for the sender is that, when the token is received and validated, Sender Agent knows that MA has been correctly executed at the recipient’s host, because the correct token that has been recovered means that MA has been given the correct resources at the recipient’s platform (which in this case the recipient’s private key). Thus, the access to the
resources at the recipient’s host is not denied. In this case, Recipient Agent must provide its private key to decrypt Cipherkey and sign \( T \).

For Recipient Agent on the other hand, it does not have to be burdened with the details of the decryption process of the plaintext. It is only required to provide its private key and authenticate MA. It can verify that MA indeed comes from the sender’s host by checking the signature and the integrity of the agent’s code (by verifying both \( S \) and \( H(Cd) \)). If both are valid, then the message and agent indeed come from the trusted sender. In addition, it can check whether the plaintext is not tampered by calculating a new hash code, and comparing it with the one received from the agent.

### 5.4 MAgSeM Communication Architecture

Figure 5-6 illustrates the agent communication architecture in MAgSeM. The rest of this section describes the components of the architecture as well as the actions of each agent in details.

#### 5.4.1 Certificates and Keys

Certificates are needed in order to authenticate each other. SSL channel can be established between two hosts by exchanging certificates. In every host, a self-signed certificate is created beforehand. The private key is kept in a Keystore at the host and the public key (bind with a certificate) is exchanged with other users as well as kept in the server\(^{45}\). Other hosts’ certificates are kept in the Truststore.

\(^{45}\) As described in footnote in Section 5.3, we assume that all certificates have been exchanged beforehand.
5.4.2 Message Format

All communications and message exchange in MAgSeM followed Agent Communication language Specification (FIPA-ACL), where performatives \(^{46}\) are provided to ease the communications processes. MAgSeM uses performatives like INFORM and REQUEST among the agents during the communication processes.

5.4.3 Different Agents Actions

1. **Interface Agent (IA)**

   IA provides Graphical User Interfaces (GUI) for authentication and message purposes. For authentication:
   
   1. takes the user ID and password, as well as the user’s IP address, and sends it to SvA to be authenticated
   2. updates the list of IP addresses of other users from the server in a periodic manner.

---

\(^{46}\) Refer Section 4.2.4
When the user has finished writing a message, IA:
3. retrieves the recipient(s) names and the message from the interface
4. sends INFORM message and both information to DOA, and waits its confirmation of received message
5. waits for INFORM message from MTA, and displays a message to sender regarding success or failure of sending the message.

2. Data Organizer Agent (DOA)
When DOA receives the message from IA, it does the following:
1. Splits the message into Recipient(s) and the actual message.
2. Saves the actual message into a file (plaintext), which will later be retrieved by MTA.
3. Sends INFORM message and the recipient(s) name to MTA
4. sends INFORM message to IA

3. Multi-tasking Agent (MTA)
MTA receives the recipient(s) name from DOA, and for each recipient:
1. sends REQUEST message to each recipient’s host (to the recipient’s CLA) to send a message
2. If the recipient is ready to accept the message, CLA will send an answer message containing ‘Agree’, a name of available RA, and the agreed mlc for K2, which is shared between the sender and recipient. Otherwise, a ‘Reject’ is sent.
3. Then, MTA sends INFORM message, with the recipients’ address, RA’s name, and mlc specification for K2 to cA
4. If there is any undelivered message notification from cA, MTA will make a new request to the affected recipient after 60 seconds\(^\text{47}\). The undeliverable condition may due to a sudden unavailability of the recipient’s agents, or disconnected system, or

\(^{47}\) The \textit{retry} time to resend the message can be determined by users. For the ease of implementation, we will set it into 60 seconds. In real practice, setting it to 60 seconds may result into bottleneck at the recipient’s machine because the time interval is too short.
because the number of connections have exceed the maximum value, or because the recipient found that the message is modified and not valid.

5. Sends INFORM message to DOA
6. Sends INFORM message to IA regarding success or failure of sending the message.

4. **Communication Listener Agent (CLA)**

CLA listens to any incoming request to send message from other MTAs. When a new request is received:

1. CLA checks the number of available RAs used at the moment. If the numbers of RAs are less than the maximum numbers allowed at a time$^{48}$, then RA accepts the request with an ‘Agree’ answer, otherwise a ‘Reject’.
2. Calculates com$_\text{layer}$ for the communication, and determine $mlc$ from the MLC specification for K2. For Layer 2 communication on mobile device, the sender agent who makes the request will tell CLA, that it is using mobile device in the ACL message$^{49}$.
3. Creates an instance of RA, which will be responsible for entertaining any incoming MA carrying the sender’s message
4. Sends the RA’s name and $mlc$ to the MTA that made the request if the answer is ‘Agree’, otherwise sends ‘Reject’.

5. **Crypto Agent (cA)**

When cA receives a message from MTA, it will do the following:

1. Determines com$_\text{layer}$ between sender and recipient based on their L0S
2. Based on com$_\text{layer}$, cA chooses $mlc$ for K1 from the MLC specifications
3. Then, it creates an instance of SUA that will prepare the plaintext using cryptography protocols.

---

$^{48}$ In MAgSeM, a user can set up a maximum number of connections it wants to receive at a time, in order to control the use of CPU processing power, which can affect the system performance

$^{49}$ CLA instantiates a special version types of RA to deal with agents that reside on mobile devices. Refer to Section 6.3.2 for more details.
4. When creating SUA, the parameters that are passed to SUA are the RA’s name and address, mlcs for K1 and K2, keystore that keeps the private key, and truststore that keeps the recipient’s certificate.
5. cA then monitors SUAs, and the flow of the message, whether, the message is delivered properly, and if not, it gives the undelivered address to MTA.

6. **SetUp Agent (SUA)**

   1. An instance of SUA is created by cA, which is responsible for preparing the message and applying appropriate security mechanisms such as discussed in Section 5.3.4.1\(^{50}\)
   2. SUA waits for MA to send signed $T$
   3. When $T$ is received, SUA verifies that $T$ is valid, and then $hashKey$ is sent to MA.
   4. When receiving INFORM message from MA, SUA checks whether its content is ‘Finish’ or ‘Reject’. If it is ‘Finish, SUA sends INFORM message to cA with ‘Finish’ content, or otherwise ‘Reject’, so that cA can ask MTA to resend another request to Recipient.

7. **Mobile Agent (MA)**

   Steps taken by MA are the same such as discussed in Section 5.3.4.2.

   1. Once arrived at the recipient’s host, MA makes a REQUEST to RA to process the message it carries
   2. Waits for a result from RA regarding $Cd$ and $S$.
   3. If ‘Invalid’ message is received, reports to SUA and terminates. If both are valid, makes a REQUEST to sign $T$
   4. Sends signed $T$ to SUA and waits for $hashKey$.
   5. Once $hashKey$ is received, makes a REQUEST to the Agent management System (AMS) of the recipient’s host to create a new agent (DA)\(^{51}\) for the decryption processes with $hashKey$ and the ciphertext passed as arguments.

---

\(^{50}\) In Section 5.3.4.1, Sender Agent is now replaced by SUA

\(^{51}\) Cd in Section 5.3.4.4 is replaced by DA
6. When DA has finished decrypting the ciphertext, it sends INFORM message together with ‘Finish’/’Reject’ as its content to MA
7. If ‘Finish’ is received, sends INFORM message with ‘Finish’ content to SUA, and terminates itself. If ‘Reject’ is received, that means, DA has not successfully decrypted the message. MA sends an INFORM message with ‘Reject’ content to SUA, and then terminates itself.

8. **Receiver Agent (RA)**
   1. RA waits for any REQUEST to process messages from MA.
   2. RA does the security steps taken such as discussed in Section 5.3.4.3
   3. RA signs $T$, when REQUEST message is made from MA.
   4. When the plaintext $P$ and $H(P)$ are received from DA, recalculate $H(P)$ from $P$ and check if $P$ is tampered.
   8. Sends INFORM message to DA whether $P$ is ‘Valid’/’Invalid’
   5. If $P$ is valid, put it in the Message queue and notify the user.
      If $P$ is invalid, INFORM DA and remove $P$
   6. Sends INFORM message to the user and terminate itself.

9. **Decrypt Agent (DA)**
   1. Once executed, DA performs security steps such as discussed in Section 5.3.4.4
   2. After decrypting ciphertext, DA sends INFORM message to RA about $P$ and $H(P)$, so that RA can recalculate the hash and validate the plaintext.
   9. Receive ‘Valid’/’Invalid’ message from RA
   10. If ‘Valid’ message is received, sends INFORM message with ‘Finish’ content to MA to inform that the decryption process is a success and it has finished the decryption process. Otherwise, sends INFORM message with ‘Reject’ content to MA.
   3. Terminates itself.
10. Server Agent (SvA)

SvA works at a server and manages certificates and users’ information. It waits for any REQUEST to authenticate a user form an IA. SvA also manages lists of users including their names, roles, L0s, and IP addresses. This list will be updated periodically to get current IP address of other users.

SvA’s actions include:

1. Waits for REQUESTs for authentication.
2. Takes the user ID and password and matches it with the one in the database
3. If both are matched, keeps the IP address in the database
4. Sends INFORM message to IA
5. Waits for REQUESTs for updating IP address
6. Generates a list of recipient’s with their current IP addresses from the database and sends it to the agent that made the request

5.5 SUMMARY

This chapter has presented an architecture based on MAS to secure online communications between two nodes. MAS is suitable for MAgSeM because of the distributed nature of the problem that can be solved with MAgSeM, where the information and expertise are distributed. For example, the sender needs information of K2 as well as the private key of the recipient which are distributed across the network.

In addition, this architecture underlines two significant properties: (1) how MLC is implemented in MAgSeM to provide appropriate security mechanisms for a particular communication, and (2) how the control of data is laid at the hand of the sender, which is done by keeping information on the encryption key at the sender’s side.

The proposed MAgSeM provides an improved approach for online communication by providing flexible security mechanisms that can cater for different security requirements and give better control over the data to its sender (the originator of communication). MAgSeM incorporates the characteristics of the agents such as coordination and cooperation, extensible,
and mobile, which give advantages when used to cater for the security processes. This architecture is not only suitable for e-health, but also for other domains that involve distributed processes.

The next chapter discusses a design and implementation of MAgSeM-based system as well as the non-agent based system, followed by the experimentation, evaluation and results presented in Chapter 7.
CHAPTER 6  THE MAgSeM SYSTEM

6.1 Introduction

This chapter presents the design and implementation of agent-based system and non-agent based system that we have developed for a proof-of-concept in this research. The agent-based system and the non-agent based system will here after be called ‘MAgSeM-based’ and ‘Socket-based’ systems respectively. The ‘Wireless’ system refers to the system in mobile or lightweight devices, while the “Wired” system refers to the system that is implemented on PC or any machine that does not use wireless connections.

Both systems are built on the application layer and network layer\textsuperscript{52}, where we provide security by providing cryptographic protocols on the data at the application layer as well as the SSL channel on the network layer. In Section 6.2, we discuss the supporting tools that we used to develop the MAgSeM-based and Socket-based systems. In Section 6.3, we discuss in details the implementation of the MAgSeM-based system including the cryptographic classes and methods, while Section 6.4 details the implementation of the Socket-based system, and later we summarize our work in Section 6.5.

6.2 Supporting Tools

This section describes the supporting tools used in developing both MAgSeM-based and Socket-based systems, including tools to develop the agents, the plug-ins, as well as for the

\textsuperscript{52} Refer to Chapter 2, Section 2.2.1
system development in mobile or lightweight devices. We also describe the library that provides the cryptographic functionalities.

6.2.1 Java Agent Developement Environment (JADE)

6.2.1.1 Agent Platforms

An agent platform is an environment used to develop multi-agent systems. There are many platforms that have been developed for this purpose such as JADE (Bellifemine et al., 2001; Bellifemine et al., 2007), Aglet (Lange & Oshima, 1998; Lange et al., 1997), and Cougaar (Helsinger et al., 2004), to name a few.

In this research, we use JADE as our supporting tool to develop our MAgSeM-based system. JADE is a software framework that is implemented based on Java Language (Arnold & Gosling, 1998), which complies with FIPA specification. FIPA is an organization “that promotes agent-based technology and the interoperability of its standards with other technologies” (Brien & Nicol, 1998; FIPA). JADE includes a runtime environment to create agents, library of classes used to develop the agents, and graphical tools to administer and monitor the agents. JADE provides a platform for agent communications, agent management (which include creation, removal, execution, migration, finding other agents, etc), and error notifications.

JADE platform is chosen because of several advantages:

1. JADE is an open source software, which can give the developer an absolute control over the framework and it is free. All source codes can be obtained from JADE website: http://jade.tilab.com/ by signing in as a member.

2. JADE supports many plug-ins or add-on functionalities such as security support, deployment of JADE agent in mobile or lightweight devices, web services support, and other utility supports, which can also be downloaded from JADE website.

3. JADE enables agents to be run on multiple environments and devices.

4. There is a mailing list for JADE communities who seek continuous technical support and programming advice while developing JADE agent systems.
6.2.1.2 Container and Platforms

In a JADE's Platform, there are containers, where the active agents reside. In a platform, a main container must always be active and other containers may join with it. In order to join to the Main container, a container (which is non-main container) must register with it when it is first started to find and use services that are offered by the Main container.

Platforms in JADE can be connected over the network such as described in Figure 6-1, taken from (Caire, 2007). From the figure, there are two platforms (Platform 1 and Platform 2) connected through a network. The agents are labelled as $A_i$. In Platform 1, there is a Main container, with Container 1 and Container 2 register with it. An agent must know other agents’ unique names in order to communicate with each other in the same platform. (e.g.: agents $A_1$ and $A_2$). An agent must know other agents’ names and addresses for communications between agents in different platform (E.g.: $A_1$ and $A_5$). The agent communications in JADE is by default, carried out using HTTP as a message transport protocol, and thus, secure channel can...
be realized using SSL and integrated in JADE. The user can choose to change to other transport protocols such as IIOP (FIPA-IIOP, 2002) and WAP (FIPA-WAP, 2001).

Inter-platform migration is allowed in JADE. However, an additional plug-ins namely Inter-Platform Mobility Service (IPMS) is needed to support migration between different platforms. The plug-ins are discussed in the next section.

6.2.1.3 JADE plug-ins

In this research, we use JADE version 3.5, and two JADE plug-ins (or additional libraries) to develop the agent system, which are:

1. Inter-Platform Mobility Service (IPMS)

   IPMS can be downloaded from https://tao.uab.cat/ipmp/. It is used to enable inter-platform mobility in JADE using HTTP/HTTPS protocols, which is to allow a platform to accept foreign mobile agents from another platform and execute their codes. There are two versions of IPMS, which are the development version and the stable version for mobile agent migration. The development version performs better for agents with larger data sizes (larger than one megabyte). The stable version performs better for mobile agent with smaller data size (Juan, 2008).

   According to Juan (2008), there are four different types of migration strategies in IPMS development version, which are the Push Cache Transfer Protocol (PCTP), On-Demand Transfer Protocol (ODTP), REST Transfer Protocol, and Fragmented Transfer Protocol (FrTP). He added that the most suitable migration strategy to transfer mobile agents with a large data in JADE is to use FrTP\textsuperscript{53}. It transfers the agent’s code, data, as well as the agent’s state in several ACL messages, instead of one transfer in a relatively large size of an ACL message. Thus, the system performs better because bad system performance can be avoided as a result of using a large message size in one migration. In this research, we use the FrTP strategy of IPMS to migrate the mobile agent to carry a large data size.

\textsuperscript{53} For the other migration strategies, readers are referred to (Braun & Rossak, 2004; Juan, 2008)
2. JADE-Lightweight Extensible Agent Platform (JADE-Leap)

JADE-Leap is used to allow JADE agents to be deployed on mobile or lightweight devices (Moreno et al., 2003). JADE-Leap categorizes four versions of Java environments. The following describes each environment (Moreno et al., 2003):

1. **JADE-Leap (j2se):** J2SE refers to Java environment executed on the Wired systems. The JADE-LEAP (j2se) is used to execute agents on PC and servers on wired network, with JDK1.4\(^{54}\) or later.

2. **JADE-Leap (pjava):** used to execute agents on mobile devices that support J2ME with CDC\(^{55}\), which is commonly used in PDA.

3. **JADE-Leap (midp):** used to execute agents on mobile devices that support MIDP1.0\(^{56}\) or later, which is commonly used in Java enabled mobile phones.

4. **JADE-Leap (dotnet):** used to execute agents on PC and servers on the Wired network that run Microsoft .Net framework.

---

\(^{54}\) Java Development Kit (JDK), used to develop Java programs. It can be downloaded form [http://java.sun.com/javase/](http://java.sun.com/javase/)

\(^{55}\) Java 2 Platform Micro Edition (J2ME) and Connected Device Configuration (CDC), will be described in Section 6.2.2.2

\(^{56}\) Mobile Information Device Profile (MIDP), will be described in Section 6.2.2.2
These four versions of JADE-Leap can co-exist with each other, so that JADE can support different types of devices and networks. This is shown in Figure 6-2, taken from (Caire & Pieri, 2008).

6.2.1.4 Inter-agent Communications in JADE

An agent has behaviours, which represent the agent’s tasks that are assigned to it. The MAgSeM-based system uses three types of behaviours that are the cyclic behaviour, generic behaviour, and one-shot behaviour. Cyclic behaviour allows the execution of behaviours to be executed forever, until the agent terminates. This type of behaviour is suitable for an agent who is waiting for a request, such as CLA and SvA. Generic behaviour operates on the current state of the agent. Different tasks are executed according to its state. Most agents in the MAgSeM-based system operate on generic behaviours. One-shot behaviour allows the execution of behaviours to be performed only one time. This type of behaviour is suitable for simple task that needs to be executed only once.
Communications between agents can be realized through behaviours. The inter-agent communications are performed by exchanging messages asynchronously among agents that are executing tasks. FIPA-ACL is used as a communication language. In this research, we use two types of performatives\(^{57}\), which are REQUEST and INFORM. Figure 6-3 shows an example of DOA sending an ACL message to MTA.

```java
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID("MTA", AID.ISLOCALNAME));
msg.setContent(recipients);
msg.setConversationId("New-Message");
myAgent.send(msg);
```

**Figure 6-3: Examples of sending an ACL message**

1. **ACLMessage class**: an object from this class is used to create an ACL message by setting up the attributes of this class through methods, such as `addReceiver()`, `setContent()`, and `setConversationId()`.
2. **Performative**: the type of performative can be specified in the first line, which is the `ACLMessage.INFORM`. The `ACLMessage.REQUEST` can be used to get the `REQUEST` performative.
3. **Receiver**: DOA sets the receiver as MTA, and this can be performed using `addReceiver()` method.
4. **Content**: The content of the message is the recipient name.
5. **setConversationId** is used as a control mechanism for receiving a reply (Bellifemine et al., 2007). For example, DOA can use the Id of the conversation to determine what kind of action that must be performed next. For instance:

   ```
   IF conversation_Id == "New Message"
   Perform Action-1
   ELSE IF conversation_Id == "New-Request"
   Perform Action-2
   ELSE
   Perform Action-3
   END-IF
   ```

6. **`myAgent.send(msg)`**: DOA sends the ACL message to MTA.

\(^{57}\) Refer to section 4.2.4
An agent must provide a template to receive an ACL message. `MessageTemplate` class provides methods of receiving ACL messages by using logic operators (and/or/not) on the `ACLMessage` class attributes. Figure 6-4 shows an example of codes that show how MTA receives an ACL message using ‘and’ operator. MTA receives the message by using `ACLMessage` attributes such as `MatchConversationId` and `MatchPerformative` as a control mechanism. Then, `blockingReceive()` method is called. It simply means that:

“Only accept a message with ConversationId equals to “New-Message” and Performative equals to `ACLMessage.INFORM’.”

Everything else is blocked or ignored. Other `ACLMessage` attributes that can be used include `MatchPerformative()`, `MatchSender()`, `MatchConversationID()`.

### 6.2.2 Tools for Mobile Devices Development

#### 6.2.2.1 J2ME, CLDC, and MIDP

Java 2 Platform Micro Edition (J2ME) is a Java platform that is used to develop applications on mobile or lightweight devices. J2ME provides a limited version of Java Application Programming Interface (API) for the application development, due to the limited resources of the mobile devices especially in memory and processing power.
Connected Limited Device Configuration (CLDC) provides the Java runtime environment for mobile application developer, specifically for devices with limited memory and processing power such as Java-enabled mobile phones. For devices with higher memory like PDA (such as more than 2MB memory), Connected Device Configuration (CDC) is used instead of CLDC.

The Mobile Information Device Profile (MIDP) is the backbone of J2ME, which is a profile specification developed to assist the development of Java application on mobile devices. MIDP lies on top of CLDC, on the J2ME protocol stack, shown in Figure 6-5, taken from Goyal (2005).

![Figure 6-5: J2ME protocol stack](image)

MIDP helps minimize the usage of memory and processing power by providing fundamental APIs to create applications on mobile devices (Goyal, 2005). For instance, basic GUI components can be developed using `javax.microedition.lcdui` package and displayed on the screen of mobile devices.
6.2.2.2 Sun Java Wireless Toolkit (WTK)

WTK is an Integrated Development Environment (IDE) to develop mobile applications. It provides environments for compiling, pre-verifying classes, packaging .jar files, and running applications through an emulator that are based on CLDC and MIDP. It can be downloaded from http://java.sun.com/products/sjwtoolkit/download.html.

Applications that run on mobile devices are called MIDlets. A MIDlet can be simulated using the emulator on WTK before being deployed on the real mobile device. Two files are needed in order to deploy a MIDlet on a mobile device, which are of .jar and .jad\(^{58}\) extensions. The MIDlet is packaged inside the .jar file, together with all resources such as Java classes, input data, as well as libraries to run the MIDlet. The .jad file contains the description of the .jar file, which is required for the deployment purposes.

6.2.2.3 Obfuscation

In the programming field, obfuscation is a process to scramble the programming code into some forms that is harder to read. In Java, obfuscation is a process to scramble the .class file, in order to protect the code so that it is harder to decompile it back to .java file program. The advantages of using obfuscation are to protect the program from being read by a third party, reduce the size of the application, and improve the performance of the application runtime Tyma (2003) and White (2005). In J2ME, obfuscation can reduce the size of the .jar file to a smaller size, which is convenience to be deployed in lightweight devices.

We use ProGuard (http://proguard.sourceforge.net/); an obfuscator purposely created for Java program. ProGuard can be integrated with WTK, and used to automate the processes on Java class files. The processes consist of shrinking, optimising, obfuscating and pre-verifying the classes. Firstly, ProGuard detects and removes unused classes, field, attributes, and methods. Secondly, it renames the remaining classes, fields, attributes, and methods with meaningless names so that it is harder to read. Lastly, it pre-verifies the code using J2ME libraries.

\(^{58}\) Java Application Descriptor
6.2.2.4 Deploying Agents in Mobile Devices

To allow an agent on a wireless mobile device to communicate with an agent on a wired PC, such as in the situation of Paramedic at an accident spot and SCat the hospital, discussed in Section 3.2.1, we need Jade-Leap (midp) to run on the mobile device and Jade-Leap (j2se) to run on the PC. The execution of the two different environments is performed through the split execution mode, which is mandatory for mobile devices (Bellifemine et al., 2007), shown in Figure 6-6.

In the split execution mode, the container is split into Front-end and Back-end. In this case, the Back-end is a laptop, and the Main container is a PC. Front-end is actually a thin layer that provides agent with features like a normal container. However, most of the implementation of the features is done by Back-end, because of the limited resources of the mobile device.

For an agent on mobile device (node A) to run, it has to be deployed through Back-end (node B), which is then joining the Main container (node C). All communications from Front-end with the Main container must go through Back-end. Note that Back-end is not necessarily has to be another machine, sitting in the middle of Front-end and Main container. Front-end is
allowed to create Back-end when joining the Main container. In other words, the Main container can also function as Back-end.

### 6.2.3 Cryptographic library: Bouncy Castle

In Java, the implementation of the functionalities of the cryptography protocols is basically based on Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE) (Hook, 2005). JCA and JCE provide classes and interfaces needed to implement the cryptography protocols. We use Bouncy Castle (BC) Provider to provide JCA and JCE functionalities. The BC libraries can be downloaded from [http://www.bouncycastle.org/](http://www.bouncycastle.org/) to provide cryptography protocols to support functionalities in our proposed security mechanism, described in Section 5.3.4. BC library provides various algorithms for Java programs, for symmetric and asymmetric key generations, encryption, decryption, signature, and hash messages. In addition, BC provides cryptography library for lightweight devices.

The BC library also provides binary encodings, for example **Base64** or **Hex**. The encoding process is required to encode binary data, into a base 64 (Base64) or base 16 (Hex) representations. The encoding process is commonly used when the binary data needs to be transferred over a network (wired or wireless) on different operating systems, and to make sure that the data received will preserve the exact same format as it is before transmission.

We have so far discussed the supporting tools that are used to implement the MAgSeM-based system. Now, we continue to explain our system implementation in greater details, discussed in the next section onwards.

### 6.3 MAgSeM-based System Implementation

In this section, we discuss in detail the implementation of our MAgSeM-based system, which include examples of communication protocols as well as the agent classes and cryptographic methods used to develop the system.
6.3.1 Agent Interactions: Wired System

![Diagram of Agent Interactions: Wired System]

Figure 6-7: Communication protocols performed on the Wired system
Consider that we have two parties communicating between a patient and a doctor. The patient wants to send a plaintext to the doctor. We assume that both parties have exchanged certificates through a secure channel that makes the patient trusted by the doctor. The patient is represented by SUA, which then dispatches MA to the doctor’s host. At the doctor’s host, the doctor is represented by RA that is responsible to interact with the incoming MA. Figure 6-7 describes parts of the whole communications between the agents, which involve MTA, CLA, cA, SUA, MA, DA and RA. Messages among agents are described in “performative: <msg1|msg2|msgn>” format.

Consider that MTA has accepted an ‘Agree’ message from Doctor’s CLA. MTA tells cA about a new message to send, together with the recipients’ address, RA’s name, and mlc specification for K2. cA then determines the com_layer value between the patient and the doctor based on their L0s (which is Layer 1). Based on com_layer, cA determines K2 and creates an instance of SUA. After applying appropriate security protocols, SUA creates and dispatches an instance of MA to carry the data to the doctor’s host.

Once arrived at the doctor’s host, MA initiates a communication with RA. MA makes a request to RA to process the message it carries. RA then processes the message and informs MA whether the code (DA’s code) MA is carrying on the message is valid or not to be executed. If the code is valid, MA retrieves Token, and requests RA to sign it. Once signed, MA sends the signed Token back to SUA. SUA processes the Token, and if it is not tampered, SUA sends hashKey for the decryption process to MA. Once received, MA makes a request to execute DA with hashKey as an argument. DA performs the decryption process to recover P. When P is recovered, DA sends P and H(P) to RA for verification. RA informs DA for the verification result. DA reports the result to MA, and finally, MA informs SUA and terminates.

From the communication shown in Figure 6-7, we observe that each agent sends a message such as an INFORM message with its content (such as ‘Valid’/’Invalid’) to report the status of the current job or task. This result/partial result is important to the other agents to determine
their next steps. The INFORM message is also a way of reporting that a task is done, from lower layer agent to the higher layer agent\textsuperscript{59}.

### 6.3.2 Agent Interactions: Wireless System

As we mentioned in Section 6.2.2.4, the split execution mode is needed in order to run agent on mobile devices. However, the split execution mode \textit{does not support agent mobility}. For example, JADE does not support agent migration from a mobile device to a PC. This is because JADE-Leap (midp) does not support the IPMS plug-in. However, we believe that this will be JADE's future work to upgrade the JADE-Leap (midp) to support mobile agent migration.

To begin with, one of the reasons the mobile agent approach is chosen in this research is that it can automate the security processes at the recipient side without intervention of the user. Therefore, to make up what we are lacking from the MA’s part, we exploit the agent’s communication feature to handle the security processes described in Section 5.3.4, without using the mobility characteristics. We modify the MAgSeM-based Wired system to be suitable to be performed on the MAgSeM-based Wireless system on mobile devices like the following:

1. we start the agent by joining it to the main container of the recipient,
2. the interfaces are provided by midp components,
3. only one agent is operated on the mobile device, where we modify the SUA’s code to perform MTA’s tasks,
4. we modify DA’s code to perform MA’s tasks at the recipient’s side
5. the recipient will be represented by RA, which is slightly modified to adapt with the processes received from different environment, i.e. mobile devices.
6. we call these agents as specialized SUA (sSUA), specialized DA (sDA), and specialized RA (sRA).

\textsuperscript{59} Refer Figure 5-2
We use Layer 2 for communications security on mobile devices\(^{60}\). Like the Wired system, the DA’s code, is still wrapped in a .jar file, and is executed once its validity is verified at the recipient’s side. Figure 6-8 shows a step-by-step communication processes for the specialized agents.

\(^{60}\) Refer to Section 3.2.2.3

---

**Figure 6-8: Communication protocols performed on Wireless to Wired systems**
Consider that a paramedic at an accident spot, contacting SC at the hospital. The paramedic is represented by sSUA, sends request to send a message to the SC’s CLA. CLA notices that the request is from an agent on a mobile device, because the request contains ‘Mobile’ in the message. If CLA agrees to receive the message, it instantiates sRA, and then informs sSUA about the sRA’s name together with the mlc specification for $K_2$. Then, sSUA performs the security protocols on the data such as described in Section 5.3.4.1, before sending it to sRA.

sRA performs processes such as described in 5.3.4.3. Then the verification result of the code (either Valid or Invalid) is sent back to sSUA. If the result is valid, sSUA requests to create an agent (which is sDA). sRA then asks the platform to instantiate sDA. Upon instantiated, sDA requests sRA to sign $Token$, and then send it to sSUA. sSUA verifies $Token$ and sends $hashKey$ to sDA. sDA performs the decryption process to recover $P$. Afterwards, sDA sends $P$ to sRA for validation purposes, and reports the result back to sSUA.

### 6.3.3 Agent Base Classes Implementation

We revisit Section 5.3.3, and show our implementation of the agent classes in the MAgSeM-based system.

#### 6.3.3.1 Certificates

Certificates in the MAgSeM-based system are created using ‘keytool’ command that can be found in the bin directory of the JDK distribution. keytool is a basic command line option, which can be used to generate a keystore (which are used to store the keys and certificates). keytool also allows importing and exporting X.509 certificates (Housley et al., 2002) to and from the keystore.

It is assumed that a security administrator has already created a certificate for every user, and certificate exchanges are done through a secure channel. A certificate file is stored in the server with the following format:

```
<Name-Role-L0>
```
For example, a certificate file of *John-Doctor-1.cer* is created for a doctor named John, which describes a certificate for a doctor named John, with $L_0$ equal to Layer 1. From the certificate, SvA extracts the information to form a list of users with their role and default layers. The IP address will be added later when a user logs into the MAgSeM-based system.

### 6.3.3.2 Interacting with users

![Interface for user authentication](image1.png)

**Figure 6-9: Interface for user authentication**

![Interface for message composition](image2.png)

**Figure 6-10: Interface for message composition**

For the Wired system, IA provides GUI for user authentication (Figure 6-9) and Message composition (Figure 6-10). IA gets the user ID and password and sends them to SvA to be authenticated, along with the user’s IP address. If the ID and password match, SvA will keep record on the IP address, which is to be used by other users. SvA also sends a list of IP addresses of the other users. This list will be updated periodically to get current IP address of
other users. The recipient’s list is stored in the format of <Name, Role, L0, address>, which then stored in a file named Recipient.txt. Figure 6-11 shows an example of a list of recipients.

```
Alex,Doctor,1, 137.92.31.166
Amin,Param,2, 192.168.1.118
Bob,Patient,1,192.168.1.5
Dory2,SW,3, 137.92.31.156
Dylan,SA,4, 137.92.31.133
John,Doctor,1, 137.92.31.165
Jenny,Nurse,1, 137.92.31.177
...
```

Figure 6-11: List of recipients

IA provides interfaces to allow a user to compose text messages and let him/her choose the intended recipient(s). Then, IA forwards the recipient’s name and the message to DOA in a string representation separated with symbol “|” like the following:

```
<recipient-name | messages>
```

Figure 6-12: Interfaces for the MAgSeM-based Wireless system
For Wireless system, the GUI is provided by the midp components. Figure 6-12 shows some of the interfaces on Wireless system run on emulator. All data entered by the user will be fetched by the agent, which later applies appropriate security mechanisms to the data.

6.3.3.3 Sending Request to the other Platform

For Wired system, DOA receives the message form IA, and splits the messages back into a list of recipients and the actual message into plaintext. A function `splitString(String s, String delimiter, int nom)` is called, to split a string `s` with “|” type of delimiter, into fragments stated by `nom`. After that, the list of recipients is forwarded to MTA.

```java
for (int i = 0; i < num; i++){
    String address = fr.getAddress(recipients[i]);
    address_array[i] = address;
    AID aid = new AID();
    aid.setName("CLA"+address+":1099/JADE");
    aid.addAddresses("http://"+address+":7778/acc");
    ACLMessage msg1 = new ACLMessage(ACLMessage.REQUEST);
    msg1.setConversationId("Send-MSG");
    msg1.addReceiver(aid);
    send(msg1);
}
```

Figure 6-13: MTA sending a request to every recipient’s CLA

Figure 6-13 shows a fragment of codes that performs process to send a request to CLA. `fr` is an object created to read and get the recipient’s address from the recipient’s list. For each of the recipient, MTA sends a REQUEST to send a message to the recipient’s CLA. MTA takes each of the recipient address and assigns it to an AID object, with `setConversationId` set to “Send-MSG”.

If the recipient is ready to accept the message, MTA will receive an answer message like the following format: `<Agree|RA-name|mlc>`, which indicates that the recipient’s host is

---

61 The full codes of `splitString()` can be found in Appendix A
62 AID is a class to construct an agent’s identifier, which we can set the name and address
ready to receive a message; the agent that will entertain the message is RA-name; and $K_2$ used at the sender’s side must be set according to mlc.

For the Wireless system, the agent that makes a request to the recipient’s CLA, must set the content of the message to be “Mobile”, using `setContent("Mobile")` so that the recipient knows that it will deal with a user that operates on a mobile device.

### 6.3.3.4 Identify com_layer

```java
public int calculateCM(int dlsender, String recipient){
    String dfFor_r = (String)determineComLayer.get(recipient);
    int rInt = Integer.parseInt(dfFor_r);
    if(dlsender == rInt)
        com_layer = rInt;
    else if (dlsender < rInt)
        com_layer = rInt;
    else
        com_layer = dlsender;
    return com_layer;
}
```

#### Figure 6-14: Determining com_layer

Figure 6-14 shows `calculateCM()` method, which is to calculate com_layer for a particular communication. It uses two parameters that are dlsender, which is the $L_0$ for sender, and the recipient’s name. First it obtains the $L_0$ for the recipient, `dfFor_r`, and then calculates the com_layer according the rule specified in Section 5.3.1.

For the MAgSeM-based Wireless system, com_layer does not have to be calculated. Once the recipient discovered that it will deal with a mobile user, the com_layer will automatically be set to Layer 2.

### 6.3.3.5 Storing MLC specification

In Section 5.3.2, we have discussed that the MLC specification is stored in the following format: `<Algorithm-lengths-mode-padding>`. We implement the specification in a class called
MLCSpec. The specification is stored as arrays of strings, containing information about the selected algorithms as shown in Figure 6-15.

```java
String Layer1[] = {"AES-256-CBC-P7","BLO-256-CBC-7"};
String Layer2[] = {"AES-192-CBC-7","BLO-184-CBC-7"};
String Layer3[] = {"AES-128-CBC-7","BLO-128-CBC-7"};
String Layer4[] = {"BLO-96-CBC-7","BLO-104-CBC-7"};
```

Figure 6-15: Implementation of MLC Specification

MAgSeM allows any types of symmetric algorithms that were proven to be reliable by experts to used, as long as it complies with the MLC specification. From the example, we choose AES and Blowfish that have a variation of key lengths to represent every layer. The encryption mode is cipher-block chaining mode (CBC) and padded with PKCS7 padding. The encryption algorithm will be selected randomly by the agent, after the com_layer has been calculated as part of the communication process.

As highlighted in Section 5.3.2, the method of selecting the algorithms in MAgSeM is more flexible compared to the method in the current technologies discussed in Section 2.6. MAgSeM offers different security levels for different types of communications, without having to reconfigure the security setting. For each communication for a sender and a recipient, the com_layer for both parties is calculated, to determine the security mechanisms.

6.3.3.6 An Example of Sending Partial Results

Sending partial result between agents is an important process in the MAgSeM-based, as it helps determine the next action that should be performed by an agent. Figure 6-16 describes codes used by cA to send the end result of a particular SUA that dispatches MA to a recipient’s platform. If the plaintext has been successfully discovered, cA will get a “Finish” or otherwise “Reject” message from SUA. This result will then be forwarded to MTA, so that
MTA can determine its next action, shown in Figure 6-17. If “Finish” is received, it will remove the recipient’s address from the recipient-to-send’s list. Otherwise, if “reject” is received, it will wait for 60 seconds\(^{63}\), before resending a request to the affected recipient.

```
ACLMessage ans = new ACLMessage(ACLMessage.INFORM);
ans.addReceiver(new AID("MTA", AID.ISLOCALNAME));
ans.setConversationId("Result-Encrypting");

if (result[0].equals("Finish"))
    ans.setContent("Finish|"+result[2]);
    //result[2]: SUA’s name
else{
    System.out.println("Cannot Deliver The Message to the following address: "+add[1]);
    ans.setContent("Reject|"+add[1]);
}
myAgent.send(ans);
```

**Figure 6-16:** cA sends partial results to MTA

\(^{63}\) As discussed in Section 5.4.3, the retry time to resend the message can be determined by users. For ease of implementation, we will set it to 60 seconds. In real practice, setting it to 60 seconds may result into bottleneck at the recipient’s machine because the time interval is too short.
if(ans[0].equals("Finish")){
    //remove of SUA from the list
    int locationIndex = ls.indexOf(ans[1]);
    ls.remove(locationIndex);
} else{
    //ans[1]:address that cannot be delivered
    //RETRY_TIME = 60 seconds
    myAgent.doWait(RETRY_TIME);
    AID aid = new AID();
    aid.setName("CLA@"+ans[1]+":1099/JADE");
    aid.addAddresses("http://"+ans[1]+":7778/acc");
    ACLMessage msg1 = new ACLMessage(ACLMessage.REQUEST);
    msg1.setConversationId("Send-MSG1");
    msg1.addReceiver(aid);
    send(msg1);

    //template to receive answer
    MessageTemplate mt =
    MessageTemplate.and(MessageTemplate.MatchConversationId("Send-MSG1"),
    MessageTemplate.MatchPerformative(ACLMessage.REQUEST));
    ACLMessage msg = blockingReceive(mt);
    if(!(msg.getContent().equals("null"))){
        splitMe sm1 = new splitMe();
        String[] result1 = sm1.splitString(msgg.getContent(),"\|",1);
        if(result1[0].equals("AGREE")){
            get the recipient's address
            String[] result =
            sm1.splitString(msg.getSender().getName(),":",1);
            String[] address = sm1.splitString(result[0],"@",1);

            String testname;
            do{
                testname = "SUA"+listsua;
                listsua++;
            }while(!(ls.indexOf(testname)<0));

            String givetocrypto =address[1]+"|"+testname+"|"+result1[1];
            ACLMessage msg11 = new ACLMessage(ACLMessage.INFORM);
            msg11.addReceiver(new AID("cryptoAgent", AID.ISLOCALNAME));
            msg11.setConversationId("Add-Not-Delivered");
            msg11.setContent(givetocrypto);
            myAgent.send(msg11);
        }
    }
}
6.3.3.7 Cryptography protocols

The cryptography protocols in MAgSeM that are detailed in Section 5.3.4 are realized by the following processes:

1. Generating a symmetric key
   a. Wired system

   ```java
   Cipher keyAes, aesCipher;
   byte[] aesKey = null;
   byte[] KeyBytes1 = null, KeyBytes2 = null;
   String algo;
   int keylength;

   public void generateSymKey(String mlc, String mlcR){
       //split mlc into algo-keylength-mode-padding for K1
       String[] tempor1 =null;
       tempor1 = mlc.split('-');
       int indexs1 = mlc.indexOf('-');
       algo1 = mlc.substring(0,indexs1);
       String kmp = mlc.substring(indexs1+1);
       String[] tempor2 =null;
       tempor2 = kmp.split('-');
       int indexs2 = kmp.indexOf('-');
       String keylength1 = kmp.substring(0,indexs2);
       String mp = kmp.substring(indexs2+1);
       String[] tempor3 =null;
       tempor3 = mp.split('-');
       int indexs3 = mp.indexOf('-');
       String mode = mp.substring(0,indexs3);
       String padding = mp.substring(indexs3+1);

       //split mlcR (from recipient) into algo-keylength-mode-padding for K2
       if (algo1.equals(algo2) && (keylength1==keylength2)){
           if(algo1.equals("AES")){
               if(keylength1 ==256){
                   try{
                       KeyGenerator kg = KeyGenerator.getInstance("AES");
                       KeyGenerator kg2 = KeyGenerator.getInstance("AES");
                       SecureRandom random = new SecureRandom();
                       SecureRandom random2 = new SecureRandom();
                       kg.init(256,random);
                       kg2.init(256,random);
                       SecretKey skey = kg.generateKey();
                       SecretKey skey2 = kg2.generateKey();
                       keyBytes1 = skey.getEncoded();
                       keyBytes2 = skey.getEncoded();
                   }catch (NoSuchAlgorithmException e) {System.out.println(e);}
                   keyAes = new SecretKeySpec(keyBytes1, "AES");
                   keyAes2 = new SecretKeySpec(keyBytes11, "AES");
               }
           }
       }
   }
   ```
The code fragment shown in Figure 6-18, is for generating symmetric keys $K_1$ and $K_2$. The generation of these keys requires mlc specifications, which are specified in mlc for $K_1$ and mlcR (obtained from the recipient) for $K_2$. From these specifications, we extract information for both keys, such as the algorithm used, key length, mode, and padding. We use KeyGenerator class to create instances of the keys using AES algorithm. Then, the keys are generated using the SecretKeySpec class. This class uses random bytes, keyBytes1 and keyBytes2, which are derived from the SecretKey class, and used them in the SecretKeySpec with the types of “AES”. Finally, the ciphers for $K_1$ and $K_2$ (aesCipher and aesCipher2) are created and initialized with AES algorithm, CBC mode, and PKCS7Padding.

b. Wireless system

For the Wireless system, we adapt the codes from (Yuan, 2003). Figure 6-19 shows part of codes that is used to generate a symmetric key. In the code, the cipher is created with the
type of Blowfish. For the key length, we use 184-bit of key (which is 23 bytes) that is assigned to a parameter named key.

```java
private KeyParameter key;
private BufferedBlockCipher cipher;
...
cipher = new PaddedBlockCipher(
    new CBCBlockCipher( new BlowfishEngine() ) );

SecureRandom sr1 = new SecureRandom();
byte[] b = new byte [23];
sr1.nextBytes(b);
key = new KeyParameter(b);
```

**Figure 6-19: Generate a symmetric key for the Wireless system**

2. Certificate and keys
   a. Wired system

```java
try{
    FileInputStream keystore = new FileInputStream(fileKS);
    ks.load(keystore, password);
    fks.close();

    FileInputStream truststore = new FileInputStream(fileTS);
    ts.load(truststore, passwordts);
    fts.close();
}
catch(Exception e){e.printStackTrace();}

X509Certificate recCert = (X509Certificate)ts.getCertificate(recipient);
recPubKey = recCert.getPublicKey();
char[] pwd = pass1.toCharArray();
Key myKey = ks.getKey(myName,pwd);//name of the keystore’s owner
senderPriKey = (PrivateKey) myKey;
```

**Figure 6-20: Obtaining public and private key**

Figure 6-20 shows the code fragment to retrieve the sender’s private key from the Keystore, and recipient’s public key from the Truststore. We use RSA (Rivest et al., 1978)
algorithm from the `keytool` command\textsuperscript{64} with the default key size of 1024 bits, for providing public and private keys for all users. In the `try` block, the Keystore and Truststore are loaded into the memory. Then, the recipient’s public key is retrieved from the X.509 certificate based on the recipient’s name through the `(X509Certificate)` `ts.getCertificate(recipient)` method. The private key is obtained from the Truststore using the user’s name.

**b. Wireless system**

In the J2ME environment, the asymmetric key pairs are generated offline (in the J2SE environment). The keys will first be serialized before exporting them to the J2ME environment. Yuan (2003) argued that by serializing the keys, it is easy to transport the key over the network, and in addition, it can avoid generating asymmetric key on the fly, because it is extremely time consuming to perform asymmetric key generation in J2ME environment.

```java
ASN1InputStream aIn = new ASN1InputStream(getClass().getResourceAsStream("Alex-Doctor-1.cer"));
ASN1Sequence encodedSeq = (ASN1Sequence) aIn.readObject();

X509CertificateStructure x509 =
    X509CertificateStructure.getInstance(encodedSeq);
SubjectPublicKeyInfo pkInfo = x509.getSubjectPublicKeyInfo();

RSAPublicKeyStructure pk =
    RSAPublicKeyStructure.getInstance(pkInfo.getPublicKey());
RSAKeyParameters pubParameters = new
    RSAKeyParameters(false, pk.getModulus(), pk.getPublicKeyExponent());
```

*Figure 6-21: Public key extraction*

As Bouncy Castle library does not support key serialization, we have to implement the serialization by saving/loading all key parameters to files. First, the key components are extracted from the public or private key and then it is serialized to files. Figure 6-21 shows an example of how the public key components are extracted from Alex’s certificate.

\textsuperscript{64} Refer to Section 6.3.3.1
We use the `ASN1InputStream` object to read and load the certificate object. Then, the `SubjectPublicKeyInfo` class is used to store the information about Alex’s public key. Afterwards, the `RSAPublicKeyStructure` and `RSAKeyParameters` are used to extract the public key components that are the *modulus* and *public exponent* to the `pk` object. Figure 6-22 shows how to serialize the components into files (`RSAmod1.dat` and `RSAPubExp1.dat`).

![Figure 6-22: Key serializations](image)

```java
BigInteger mod = pk.getModulus();
out = new FileOutputStream("alex/RSAmod1.dat");
out.write(mod.toByteArray());
...

BigInteger pubExponent = pk.getPublicExponent();
out = new FileOutputStream("alex/RSAPubExp1.dat");
out.write(pubExponent.toByteArray());
...
```

### 3. Encryption

#### a. Wired system

Figure 6-23 shows a code fragment of the `encrypt()` method. The method encrypts a plaintext, which is a file specified as `in`, and the end result of ciphertext will be save as a file, specified as `out`. The first `try` block initializes the cipher to the mode of operation it is to be used, which in this case, a mode to encrypt.

The `CipherOutputStream` class is initialized with an output stream `out` and an AES cipher (`aesCipher`). This class is used to encrypt data as it is written to a specified output stream. The encryption is finally done by calling the `copy()` method. The data in a file is read by `is` and written by `os`, which we have initialized earlier. When the process of `copy()` has ended, the data has been completely encrypted and written to `out`. The same processes are applied when generating `Token` and `Ciphercode`. The whole `encrypt()` method can be found in Appendix A.

For asymmetric encryption to generate `Cipherkey` (Figure 6-24), that is to encrypt $K_2$ and $H(Cd)$ to be sent to the recipient’s side, we need the recipient’s public key (`pk`), and initialize a cipher (`pkCipher`) to it to be in an ENCRYPT mode. The key $K_2$ and $H(Cd)$, are encoded
into a byte array of data `strByte` beforehand. Then, we use the `CipherOutputStream` class for the encryption processes and the output of `Cipherkey` is stored in `out`.

```java
File in, out;
FileInputStream is = new FileInputStream(in);
CipherOutputStream os = null;
...
try{
    aesCipher.init(Cipher.ENCRYPT_MODE, keyAes, ivSpec);
} catch(InvalidAlgorithmParameterException e) {System.out.println(e);}

os = new CipherOutputStream(new FileOutputStream(out), aesCipher);
copy(is, os);

private void copy(InputStream is, OutputStream os) throws IOException {
    int i;
    byte[] b = new byte[1024];
    while((i=is.read(b))!=-1) {
        os.write(b, 0, i);
    }
}
```

**Figure 6-23: Symmetric encryption for the Wired system**

```java
PublicKey pk = null;
Cipher pkCipher = null;
...
File out;
...
byte[] strByte = Base64.encodeBase64(aesKeyHashStr.getBytes());
pkCipher.init(Cipher.ENCRYPT_MODE, pk);
CipherOutputStream os = new CipherOutputStream(new FileOutputStream(out), pkCipher);
os.write(strByte);

```

**Figure 6-24: Asymmetric encryption for the Wired system**

b. **Wireless system**

Figure 6-25 shows a code fragment to encrypt an array of bytes in a mobile device with a symmetric key cipher to produce `Token`, `Ciphertext`, and `Ciphercode`. The code shows the
Encrypting() method, which encrypts an array of data in bytes by using a cipher, initialized with key, which we previously created in Figure 6-19.

When the method is called, it will first call the getOutputSize() method to determine sufficient data space for the output buffer required for processBytes() and doFinal() methods. cipher.processBytes processes the array of bytes of data and put the output in result. The variable len, keeps track of the length of the processed bytes. The encryption process ends when the doFinal() method processes the last block in the buffer, and checked whether or not result has the appropriate data length.

```
...  
//call a function to encrypt
byte[] encrypted = Encrypting (bytedata);
...

public byte[] Encrypting(byte[] data) throws Exception {
    int size = 0;
    byte[] result=null;
    int len = 0;
    
    cipher.init(true, key);
    size = cipher.getOutputSize(data.length);
    result = new byte[size];
    
    len = cipher.processBytes(data,0,data.length,result,0);
    
    if(data == null || data.length == 0)
        return new byte[0];
    
    //process the last block in the buffer
    len += cipher.doFinal(result,len);
    
    if(len < size){
        byte[] tmp = new byte[len];
        System.arraycopy(result,0,tmp,0,len);
        result = tmp;
    }
}
```

**Figure 6-25: Symmetric encryption for the Wireless system**

Cipherkey is created using the RSA asymmetric key, which is realized in a method called RSAEncrypt(), shown in Figure 6-26. The method receives data in the form of an array of bytes and the public key to encrypt the data. The asymmetric key cipher is instantiated and
initialized with the type of RSA using the public key, and true value to indicate an ENCRYPT mode. Then, the encryption process is done using the processBlock() method with the result is finally returned to the caller function.

```java
public byte[] RSAEncrypt(byte[] toEncrypt, RSAKeyParameters pubkey) throws Exception {
    if (pubkey == null)
        throw new Exception("Generate RSA keys first!");

    AsymmetricBlockCipher eng = new RSAEngine();
    eng = new PKCS1Encoding(eng);
    eng.init(true, pubkey);
    return eng.processBlock(toEncrypt, 0, toEncrypt.length);
}
```

**Figure 6-26: Asymmetric encryption for the Wireless system**

4. Binary Encoding/Decoding

   a. Wired system

   As explained in 6.2.3, we use binary encoding and decoding to transfer binary data over a network (wired or wireless) on different operating systems, so that the data will preserve the exact same format as it is before transmission. We use the Bouncy Castle library to encode and decode the binary data, such as shown in Figure 6-27 and Figure 6-28 respectively.

   ```java
   byte[] ciphertext;
   String s = new String(Base64.encodeBase64(ciphertext));
   ```

   **Figure 6-27: Binary encoding for the Wired system**

   ```java
   String ciphertext;
   byte[] ciphertextByte =
       Base64.decodeBase64(ciphertext.getBytes());
   ```

   **Figure 6-28: Binary decoding for the Wired system**

   The data that is to be encoded is in the form of an array of bytes. The function Base64.encodeBase64 returns a string as an output. The reverse is performed on the decoding process, which takes a string and decoded it back to an array of bytes.
b. **Wireless system**

For Wireless system, we use Haustein (2003)’s Base64 encoding java class to encode and decode the data, shown in Figure 6-29 and Figure 6-30 respectively. The processes to encode and decode here are similar to the one in the Wired system.

```java
byte[] strCipherkeyByte = strCipherkey.getBytes();
cipherkey = new String (bs.encode(byteCipherkey));
```

**Figure 6-29: Binary encoding for the Wireless system**

```java
String s1 = splitMessage[0];
byte[] bytecipherkey= (bs.decode(s1));
```

**Figure 6-30: Binary decoding for the Wireless system**

5. **Generating a key pair (Kp, Ks)**

We call `AsymmetricKeyMaker` class (Figure 6-31) to generate asymmetric key pairs (Kp, Ks) with RSA algorithm type for both Wired and Wireless systems. `KeyPairGenerator` class is initialized with the specified algorithm and `generateKeyPair()` method is used to generate the key pair.
The key pairs are disposable, stored by cA, and assigned to an instantiated SUA. SUA manages the keys, where it stores the secret key ($K_s$) to generate $hashKey$. The public key is embedded in DA to be used to decrypt $hashKey$ at the recipient’s side. Both $K_s$ and $K_p$ will be removed once SUA and DA are terminated.

6. Generating and Verifying Signature

```java
public class AsymmetricKeyMaker {
    public KeyPair genKey() {
        KeyPair keyPair = null;
        String algo = "RSA";
        try {
            keyPair = KeyPairGenerator.getInstance(algo).generateKeyPair();
            PublicKey kp = keyPair.getPublic();
            PrivateKey ks = keyPair.getPrivate();
        } catch (NoSuchAlgorithmException e) {} 
        return keyPair;
    }
}
```

```java
public String Signing(File code, PrivateKey senderPriKey)
    throws GeneralSecurityException, IOException{
    byte[] signCd = new byte[(int) code.length()];
    try {
        FileInputStream fis = new FileInputStream(code);
        fis.read(signCd);
    } catch (FileNotFoundException e) {...}
    catch (IOException e1){...}
    Signature sig = Signature.getInstance("SHA1withRSA");
    sig.initSign(senderPriKey);
    sig.update(signCd);
    byte[] byteSignCd = sig.sign();
    String byteSignStr = new String(Base64.encodeBase64(byteSignCd));
    return byteSignStr;
}
```

Figure 6-31: Generate key pairs

Figure 6-32: Signing with a private key
Signing() method is called to sign the agent’s code (Figure 6-32). First, the content of the code is read into byte array, signCd. Next, a Signature object (sig) is created that implements SHA1withRSA algorithm. Then, sig is initialized for signing the code with the private key of the sender (senderPriKey). signCd is then read with update() method, and finally sig.sign() is called to generate the signature. The signature of the code contained in sig as a byte array is encoded into String representation, and returned to the caller function.

```java
Signature verifySig = null;
try {
    verifySig = Signature.getInstance("SHA1withRSA");
    verifySig.initVerify(senPubKey);
} catch (NoSuchAlgorithmException noAlgorithm) {...}
...

FileInputStream f2 = new FileInputStream(input);
byte[] byt = new byte[(int) input.length()];
f2.read(byt);

boolean signatureVerified = false;
try {
    verifySig.update(byt);
    f2.close();
    FileInputStream signatureIn = new FileInputStream(signatureFile);
    byte[] signatureData = new byte[(int) signatureFile.length()];
    signatureIn.read(signatureData);
    signatureVerified = verifySig.verify(signatureData);
} catch (SignatureException signError) {...}
return signatureVerified;
```

Figure 6-33: Verifying signature with a public key

We use the VerifySignature class to verify a signature. The verification processes is quite similar to the signature generation process (Figure 6-33). A signature class is instantiated with the specified algorithm. Then, we initialize the object for verification using the sender’s public key (senPubKey). Next, the data to be verified is read using the update() method, and finally the verification process is done by calling the verify() method. If the signature is valid, the method returns true, otherwise, false. The result of the verification is stored in a boolean variable (signatureVerified).
7. Generating and Verifying Message Digest

The `ComputeMD()` method is called to generate message digest (Figure 6-34). First, we read input data into a byte array (`byteInputData`). Then, we initialize the `MessageDigest` object to implement the desired algorithm. Next, the `update()` method is called to read the data to be hashed. Finally, the hash computation is done using the `digest()` method, which returns the digest value as a byte array.

```java
public byte[] computeMD(File input){
    byte[] byteMDInputData = null;
    byte[] byteInputData = new byte[(int) input.length()];
    try {
        FileInputStream fis = new FileInputStream(input);
        fis.read(byteInputData);
        fis.close()
    } catch (Exception e) {...}
    //compute message digest
    try{
        MessageDigest md = MessageDigest.getInstance("SHA1");
        md.update(byteInputData);
        byteMDInputData = md.digest();
    }catch(NoSuchAlgorithmException e){...}
    return byteMDInputData;
}
```

**Figure 6-34: Generate message digest**

The `recomputeMD()` method is used to verify a message digest (Figure 6-35), which is used at the recipient’s side. A new message digest is computed from a file that needs to be verified, and then compared with the one received from the sender (`hashcode`). First, the file content is read into a byte array called `ByteCd`. Then, we initialize a `MessageDigest` object to implement a certain algorithm. The `update()` method is called to read `ByteCd` to be hashed. Next we compute a new hash message using the `digest()` method, and assign it to a variable named `newMsgDigest`. To test whether both `hashcode` and `newMsgDigest` are the same, we can compare it using the `equals()` method, which return `true` if it is equal, and `false` otherwise.
Figure 6-35: Recompute and verify message digest

8. Decryption

Figure 6-36 shows a code fragment of the `decrypt()` method. The steps in this method are quite similar to the `encrypt()` method (as shown in Figure 6-23). The method decrypts a ciphertext to recover the plaintext. The `try` block initializes the cipher to the mode of operation it is to be used, which is a mode to decrypt. The `CipherInputStream` class is initialized with an input stream `in`, and a cipher (`aesCipher`). The class is used to decrypt data as it is read from a specified input stream. The decryption is done by calling the `copy()` method, and the result is written to `out` as a plaintext.

```java
public boolean recomputeMD(File fstrCd, String hashcode ){
    byte[] ByteCd = new byte[(int) fstrCd.length()];
    try {
        FileInputStream fileMD = new FileInputStream(fstrCd);
        fileMD.read(ByteCd);
    } catch (Exception e) {...}

    byte[] newMsgDigest =null;
    try{
        MessageDigest md = MessageDigest.getInstance("SHA1");
        md.update(ByteCd);
        newMsgDigest = md.digest();
    }catch (NoSuchAlgorithmException e){}

    String strNewMsgDigest = new
        String(Base64.encodeBase64(newMsgDigest));
    boolean statusMD = strNewMsgDigest.equals(hashcode);
    if(!strNewMsgDigest.equals(hashcode))
        System.out.println("P has been tampered!");
    else
        System.out.println("Genuine P received.");

    return statusMD;
}
```
try{
    aesCipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
}catch(InvalidAlgorithmParameterException e){System.out.println(e);}

CipherInputStream is = new CipherInputStream(new FileInputStream(in), aesCipher);
FileOutputStream os = new FileOutputStream(out);
copy(is, os);
is.close();
os.close();
...

private void copy(InputStream is, OutputStream os) throws IOException {
    int i;
    byte[] b = new byte[1024];
    while((i=is.read(b))!=-1) {
        os.write(b, 0, i);
    }
}

byte[] decryptCiphercode = BLODecrypt(byteciphercode);
...

public byte[] BLODecrypt( byte[] data) throws Exception {
    cipher.init( false, bytekey );
    int size = cipher.getOutputSize( data.length );
    byte[] result = new byte[ size ];
    int len = cipher.processBytes( data, 0, data.length, result, 0 );
    if( data == null || data.length == 0 ){
        return new byte[0];
    }
    len += cipher.doFinal( result, len );
    if( len < size ){
        byte[] tmp = new byte[ len ];
        System.arraycopy(
            result, 0, tmp, 0, len );
        result = tmp;
    }
    return result;
}

Figure 6-36: Symmetric decryption for the Wired system

Figure 6-37: Symmetric decryption for the Wireless system
Figure 6-37 shows the decryption process for the symmetric key cipher for the Wireless system. The process is quite similar to the encryption process (as shown in Figure 6-25), except for the `cipher.init(false, bytekey)` method, where the cipher is initialized with the appropriate key bytes (`bytekey`) and is set to DECRYPT mode with the `false` value.

```java
pkCipher.init(Cipher.DECRYPT_MODE, recPriKey);
byte[] cipherkeys = new byte[(int) in.length()];
CipherInputStream is = new CipherInputStream(new FileInputStream(in), pkCipher);
is.read(cipherkeys);
```

**Figure 6-38: Asymmetric decryption for Wired system**

```java
public byte[] RSADecrypt(byte[] toDecrypt) throws Exception {
    if (caPrivateKey == null)
        throw new Exception("Generate RSA keys first!");

    AsymmetricBlockCipher eng = new RSAEngine();
    eng = new PKCS1Encoding(eng);
    eng.init(false, caPrivateKey);
    return eng.processBlock(toDecrypt, 0, toDecrypt.length);
}
```

**Figure 6-39: Asymmetric decryption for Wireless system**

Figure 6-38 shows the asymmetric decryption for the reverse asymmetric encryption (explained in Figure 6-24) to get \(K2\) and \(H(Cd)\). A private key is used to initialize a cipher (`pkCipher`) with a mode to decrypt. The `cipherkeys` represents the data to be decrypted. `CipherInputStream` class is called to perform the decryption processes and the end result will be stored in a file represented by `in`.

For the Wireless system, the asymmetric decryption process is shown in Figure 6-39. The processes are the same as in the asymmetric encryption (Figure 6-26) apart from the initialization processes in `eng.init(false, caPrivateKey)` method, where the cipher is initialize with DECRYPT mode (indicated through `false` value) and a private key.
9. SSL channel Establishment

JADE does not support an automatic switching from common channel to secure SSL channel without restarting the platform. Thus, we will have to restart our platform whenever SSL is required. We use JADE-HTTPS setting such as in Exposito et al. (2003). An example of starting a JADE platform with SSL is like the following:

```java
java jade.Boot -mtp
```

where, `machine` indicates the machine name if known, in a local network, or an IP address.

```
jade_mtp_http_https_keyStoreFile=Johnfile
jade_mtp_http_https_keyStorePass=John2010
jade_mtp_http_https_trustManagerClass=
jade_mtp_http_https_friendListFile=Johnts
jade_mtp_http_https_friendListPass=Johnts10
services=jade.core.mobility.AgentMobilityService;jade.core.migration.InterPlatformMobilityService
```

**Figure 6-40: Example of Jade.conf**

Certificate exchanges between the communicating parties are necessary to establish SSL. A sender stores certificates of the other users in a Truststore\(^65\). Then, the Truststore is stated as a parameter in the command line, in addition with the sender’s Keystore. Finally, to save time to type the command line, we can use a configuration file to store all the necessary information, and save it as a file named `jade.conf` shown in Figure 6-40 for an example of a user named John. Then, we use JADE to invoke the file using the following command line:

```java
java jade.Boot -conf jade.conf
```

\(^65\) Refer to Section 5.4.1
6.4 Socket-based System Implementation

The Socket-based system implementation is based on Java Socket programming (Sun-Java, 1995) for J2SE and J2ME environments, because it supports communication between two nodes. Java provides socket classes representing a connection between the two communicating nodes. The system implementation is divided into client-side and server side. We choose to use the term client to represent the sender, and the server to represent the recipient to describe the implementation processes. The methods for cryptographic protocols implementation is the same as the one in Section 6.3.3.6.

6.4.1 Client-side Implementation

```java
Socket socket = null;
PrintWriter out = null;
BufferedReader in = null;
try {
    socket = new Socket(ipAdd,port);
    out = new PrintWriter(socket.getOutputStream(), true);
    in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
} catch (UnknownHostException e) {
    System.err.println("host unknown");
    System.exit(1);
} catch (IOException e) {
    System.err.println("Couldn't get I/O for the connection");
    System.exit(1);
}

SSLSocketFactory socketFactory = (SSLSocketFactory)SSLSocketFactory.getDefault();
SSLSocket ssls = (SSLSocket) socketFactory.createSocket(hostname, port);
```

Figure 6-41: Creating a client-side socket

For mobile devices, the SocketConnection class is used to create a socket object (Figure 6-41). The open method of the Connector class is used to connect the socket object to the specified IP address and port. The OutputStream and InputStream objects are used to send messages to and receive messages from the server respectively.

Figure 6-42: Creating a client-side SSL-based socket
Figure 6-41 shows code fragments for creating a client-side socket. The `Socket` class is used to create a client socket, to connect to the IP address specified in the `ipAdd` and `port` fields. The `PrintWriter` and `BufferedReader` classes are used to send and receive messages from the client socket respectively. For SSL connection, we use the `SSLSocket` class to create a client socket with SSL-enabled, which will connect to the SSL-enabled socket at the server-side (Figure 6-42).

### 6.4.1.2 Creating Client Socket for Sender (Wireless system)

```java
SocketConnection sc;
InputStream is;
OutputStream os;

sc = (SocketConnection)Connector.open("socket://" + IP-add + ":"+port);
is = sc.openInputStream();
os = sc.openOutputStream();
```

**Figure 6-43: Creating a client-side socket**

For mobile devices, the `SocketConnection` class is used to create a socket object (Figure 6-43). The `open` method of the `Connector` class is used to connect the socket object to the specified IP address and port. The `OutputStream` and `InputStream` objects are used to send messages to and receive messages from the server respectively.

```java
os.write(message.getBytes(),0,message.length());
os.write(message.getBytes());
os.write("\r\n".getBytes());
os.flush();
os.close();
```

**Figure 6-44: Sending message**

If the connection is successfully opened, the client can directly use the `write` method from the `OutputStream` class to send a message (Figure 6-44). The symbol represents the end of line ("\r\n"), must be specified, to tell the server that it is the end of the message, so that the server can close the connection. Figure 6-45 shows how we read messages at the server-side.
6.4.2 Server-side Implementation

6.4.2.1 Creating a server socket for recipient

The ServerSocket class is used to create a socket on the server-side. Figure 6-46 shows the code fragment to create the socket. We give an example of how the socket is established on port 1234. After the socket is created, it listens to any connection from the client (sender). This is described in the try block. The PrintWriter and BufferedReader classes are used to send and received messages from the client socket respectively.

```java
while (true) {
    StringBuffer sb = new StringBuffer();
    int c = 0;
    while (((c = is.read()) != '
') && (c != -1)) {
        sb.append((char)c);
    }
    if (c == -1) {
        break;
    }
}
```

Figure 6-45: Receiving messages

```java
ServerSocket serverSocket = null;
try {
    serverSocket = new ServerSocket(1234);
} catch (IOException e) {
    System.err.println("Could not listen on port: 1234.");
    System.exit(1);
}

Socket clientSocket = null;
try {
    clientSocket = serverSocket.accept();
} catch (IOException e) {
    System.err.println("Accept failed.");
    System.exit(1);
}

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
```

Figure 6-46: Creating a server-side socket
For SSL connections, Figure 6-47 shows how SSL is established at the server-side on port 443. We use the `SSLSocket` class to accept any inbound connections from clients with SSL-enabled connection. The server socket is established using the `SSLServerSocket` class.

### 6.4.3 Communication Link

The Socket-based system does not operate on mobile agent or mobile code, and instead, it uses the traditional message passing method using socket connections. Figure 6-48 shows the implementation of socket connections for the Socket-based system. Client 1, which is on the Wired system connect to a server on port 1234 using a socket. Client 2 that is on the Wireless system also implements a socket to connect to the same sever, using a different type of connection.
methods. Next, the step-by-step of the communication processes in the Socket-based system are described in Figure 6-49.

The step-by-step process:

1. The server listens to any inbound connections.
2. For any client to connect to the server, it will first request for the mlc setting for K2.
3. The server then gives response with the appropriate mlc setting for the communication with the client.
4. Based on the mlc, the client applies cryptography protocols such as described in Section 5.3.4, with modifications like the following:
a. The client applies the same security protocols such as performed by the Sender Agent in Section 5.3.4.1. However, the DA’s code is replaced with a Java class in a .jar file that implements the decryption process.
b. Kp is not embedded within the class\(^6\), and instead, it will be serialized into files and read by the server.

5. The client then sends the message (Ciphertext, Ciphercode, and Cipherkey) to the server-side.
6. At the server’s side, the server performs the steps taken by RA and MA (Sections 5.3.4.2 and 5.3.4.3). Then, it signs the token, and sends it to the client to get hashKey.
7. After verifying that the token is valid, the client then sends hashKey to the server (or otherwise, the communication is ended)
8. Finally, the server un-jars the .jar file, then creates an object of the class, and calls the functions) to perform the decryption process, such as performed by DA.

### 6.4.3.1 Drawbacks

In the Socket-based system implementation, we have identified several weaknesses in using the socket-based implementation. Firstly, communications that are similar to agent-to-agent communications are not supported. In the case of the agent-to-agent communications (for the Wired system), when the plaintext has been discovered in the recipient’s platform, SUA will get a “Finish” report stating that the job is done without a hitch, or otherwise “Reject”. SUA will then report it back to cA and then cA reports it to MTA. If in case there is an undeliverable report, MTA will make another request to the recipient. These reports sends among these agents can be expected milestone of steps that have been taken, or performed by a particular agent. All of these processes are done by the agents on behalf of the user without or with minimal intervention.

\(^6\) For the MAgSeM-based system, Kp can be set as an argument when DA is first executed. The codes of DA and Java .class are quite similar and can be found in Appendix A.
However, in the Socket-based system, the user is in charge of the communication processes from start to finish. If there is an undelivered message, the user is responsible to resend the message. Thus, the user needs to be available throughout the communication processes. In other words, the security processes cannot be automated for the user.

Secondly, the server is burdened by the implementation of the Java .class object (described in Step 2) with its methods that need to be invoked to perform the decryption process. This is because, unlike agents, the Java object does not have the capability to initiate a process without being invoked by an external entity. On the contrary, an agent like DA has its own data and states and can autonomously perform tasks without interventions. As a result, the recipient does not have to be burdened by the decryption processes.

Finally, for the design and programming-wise, the code for Client 1 in Figure 6-30 is not reusable for Client 2 which have different environment. As a consequence, additional programming is needed in order to cater for the different functionalities. This is contrary to the MAgSeM-based programming, where some of the codes for agent’s communications and behaviour can be reused and modified slightly for different environments.

6.5 Summary

This chapter has discussed about the design and implementation of the MAgSeM-based and Socket-based systems. We explained the supporting tools used to build the systems. JADE platform is used to develop the MAgSeM-based system while Java socket is used for the Socket-based system. We also examined the types of tools used to develop the systems on mobile devices. Then, we discussed about the base classes and methods used to implement the systems, including the cryptographic protocols. Finally, we argued on how Socket-based system has disadvantages over MAgSeM-based system in terms of lack of security automation, problem faced by the server to handle the Java object, as well as design and programming-wise benefit.

Next chapter presents discussion and experimentation conducted on both systems including the Wired and Wireless systems on mobile devices and will evaluate the results.
CHAPTER 7 RESULT AND ANALYSIS

7.1 Introduction

In Chapter 6, we have discussed in detail the MAAsM-based and Socket-based systems implementation. In this chapter, we will discuss the experiments conducted to compare the performance of communication at every layer, including Wired and Wireless systems\(^{67}\).

In Section 7.2, we explain how we set up our experiments for both MAAsM-based and Socket-based systems. Section 7.3 describes the parameters used in the experiments for the Wired and Wireless systems. Then, Section 7.4 discusses issues that we came across while performing the experiments. The results are presented in Section 7.5. We measured and compared the communication performance in each layer for MAAsM-based and Socket-based systems in terms of (1) execution time taken for a communication transaction; which is the time for an agent to process a plaintext, to transfer the plaintext to the recipient’s side, and for the sender to get a reply from the recipient; (2) the overhead costs of the security processes. Our experiments included communication that came from Wired and Wireless systems. Finally, we present our discussion and summarize our findings.

---

\(^{67}\) As described in Section 6.0, ‘Wireless’ system refers to the system in mobile or lightweight devices, while ‘Wired’ system refers to the system that is implemented on PC or any machine that does not use wireless connections.
7.2 Environment Setup

For Wired systems, we created a controlled environment by using two PCs connected to each other in a LAN such as shown in Figure 7-1. Each computer was equipped with Windows XP professional, Pentium IV, 3 GHz CPU, and 1 GB RAM.

![Figure 7-1: Environment setup for agent and socket-based communications](image)

For Wireless systems, we used a mobile phone as the sender and a laptop as the recipient shown in Figure 7-2. We run the experiment on a mobile device and a wired laptop with the following specifications:

1. Mobile device:
   - Hewlett Packard iPaq 900, with Windows mobile 6.1, processor ARM920T PXA27x 416Mhz, and memory of 116.80 MB
   - CLDC version 1.0 and MIDP version 2.0

2. Laptop:
   - DELL Inspiron, with Windows XP Professional, processor Intel Core (TM) Duo 1.86 GHz, and memory of 1.99 GB

![Figure 7-2: Environmental setup for mobile devices](image)
7.3 Experiment Setup

The goal for the experiment is to observe the security performances in every layer in MLC, by comparing the MAgSeM-based system and Socket-based system communications. For both systems, the execution times (in milliseconds) for communications between two nodes representing sender and recipient are measured. The time starts from securing the plaintext using MLC at the sender side, until the sender receives the token back from the recipient. This is to represent a circular exchange or a two-way exchange of messages between sender and recipient. We call this two-way message exchange as Transaction Time (TransacT).

Note that the negotiation time between the sender’s MTA sending a request to the recipient’s CLA to get the information about $K2$ and RA’s name is not considered. The reason is that, if a response to the request is undelivered or rejected, MTA will have to make another request to CLA and it will consume more time and the measurement will not be consistent. Therefore, we measure the time after MTA receives an Agree message, starting from securing the plaintext.

We also measure communication efficiency with no security, which transfer plaintexts to the recipient’s side without applying any security mechanism, in order to measure the overhead cost of the security processes.

7.3.1 Wired System Communication

Figure 7-3 details the measurement of TransacT for MAgSeM-based system, starting from preparation of the data (generating Ciphertext, Cipherkey, and Ciphercode), sending the data across by MA, until the sender receives Token back from MA to ask for hashKey, using messaging from the recipient to the sender. Figure 7-4 describes the measurements for TransacT for the Socket-based system, which starts from the preparation of data (the same as in the MAgSeM-based system), sends it through a socket to the recipient, until the recipient sends the signed token back to the sender (also via socket) to ask for hashKey.
Figure 7-3: Measurement of TransacT (MAgSeM-based)

Figure 7-4: Measurement of TransacT (Socket-based)
For both systems, we conducted five experiments with plaintext sizes ranging from 1Mb to 9Mb. We designed our experiments so that, it could cover all the MLC security mechanisms\textsuperscript{68} that include data security, channel security, as well as the combination of data and channel security.

In Section 5.3.3, we have discussed that $K1$ must be kept secret at the sender host, while $K2$ is known and shared between the sender and recipient. However, for simplicity purposes, $K1$ and $K2$ are generated as the same algorithms in this experiment. In practice, the sender may use different algorithms for $K1$ and $K2$. For all data security, we choose CBC mode, and PKCS7 padding type. We use SHA1 algorithm to compute message digest and SHA1withRSA to sign the message.

\begin{table}[h]
\centering
\caption{Table 7-1: Experiment setup for wired system}
\begin{tabular}{lll}
\hline
No & Security Mechanisms & Algorithm for K1 and K2 \\
\hline
1 & Layer 1: data security and SSL channel security & AES 256-bit + SSL \\
2 & Layer 2: data security & AES 192-bit \\
3 & Layer 3: data security & AES 128-bit \\
4 & Layer 4: data security & SSL-only \\
5 & No Security Communications to transfer plaintexts & - \\
\hline
\end{tabular}
\end{table}

Four MLC security settings and one with no security were used, as shown in Table 7-1. For comparison purposes, AES algorithms with different key lengths were chosen. AES 256-bit and SSL was used to provide data and channel security for Layer 1, AES 192-bit and AES 128-bit were used to provide data security to Layer 2 and Layer 3 respectively, SSL only was used to provide channel security for Layer 4, and No Security for Layer 5. The No Security communication involves message exchanges between the sender and recipient without any security protocol applied to the communication. The SSL security was provided automatically by the JADE systems through Java SSLServerSocket object.

\textsuperscript{68} Refer to section 3.2.2.3 for MLC security mechanisms
7.3.2 Wireless System Communication

Mobile device security falls in Layer 2, for which the key lengths supported are from 112-bit to 192-bit\textsuperscript{69}. In the experiment for mobile devices, we randomly select three symmetric encryptions, with key lengths that fall within the range of Layer 2, to observe the performance executions of the algorithms with different plaintext sizes.

For both MAgSeM-based and Socket-based systems executed on mobile device environment, the execution time is calculated starting from the preparation of data (generating \textit{Ciphertext}, \textit{Cipherkey}, and \textit{Ciphercode}), sending the data across, until the sender receives \textit{Token} to ask for \textit{hashKey}.

Four experiments are conducted, with \textit{K1} is set with the setting shown in Table 7-2. The algorithms selected for \textit{K1} and \textit{K2} are different. For \textit{K1}, we provide three different algorithms that are 3DES 168-bit, AES 128-bit, and Blowfish 112-bit. The plaintext sizes used are range from 600 bytes to 200 Kb.

\begin{table}[ht]
\centering
\caption{Experiment setup for non-mobile device agent-based system}
\begin{tabular}{|c|c|c|}
\hline
\textbf{No} & \textbf{Security Mechanisms} & \textbf{Algorithm for K1} \\
\hline
1 & Data security & 3DES 168-bit \\
2 & Data security & AES 128-bit \\
3 & Data security & Blowfish 112-bit \\
4 & No security & - \\
\hline
\end{tabular}
\end{table}

For \textit{K2}, we fix the algorithms to Blowfish 184-bit. Other pre-defined parameters besides \textit{K2} are disposable key pairs (\textit{Kp}, \textit{Ks}) and recipient’s certificate or public key, which are created off-line and serialized into files.

\textsuperscript{69} Refer Section 3.2.2
7.4 Measurements

7.4.1 Time Measurement
Our system is built with Java, and therefore, we use the Java method of \texttt{System.currentTimeMillis()}, which returns current time in \textit{milliseconds}. The method is used to calculate the time intervals for the message exchanges between sender and recipient. However, the return value of this method depends on the operating system, where many of the operating systems measure time in 10 milliseconds (Sun-Microsystems, 2003). In our experiments, all experiments will be measured as an average. For example, we execute the TransacT for \(n\) times, sum them all and divide the sum by \(n\). We choose \(n = 30\), so that we could get a consistent reading of the execution times to converge at a consistent value.

7.4.2 Measurement Interval
We identify four different measurements to complete the whole communication between sender and recipient, as shown in Figure 7-5. \(n\) is the average time taken for the agent, and \(T\) is the time to complete a communication in milliseconds.

![Figure 7-5: Time intervals to complete a communication](image)

Figure 7-5: Time intervals to complete a communication
• T1 is the time taken for processing the data at the sender side, which is to generate *Ciphertext*, *Cipherkey*, and *Ciphercode*.

• T2 is the time taken, starting from sending the data to recipient, processing the token to be signed, and sending it back to the sender.

• T3 is the time taken starting from the sender’s side receives the data and processes the data, until the *signed token* is sent to the sender.

• T4 is the time taken starting from receiving *hashKey* at the recipient’s side, until the plaintext is discovered and validated.

To get TransacT and complete a circular message exchange, we calculate the following intervals:

\[ TransacT = T1 + T2 \]

By calculating TransacT, we are also able to examine the transfer time (TT), by extracting the time taken to purely transfer the message to and from the recipient as:

\[ TT = T2 - T3 \]

Note that T4 is not included in calculating TransacT, because it does not represent a circular message exchange. It starts from when *hashKey* is received until the decryption and validation of plaintext at the recipient’s side. Therefore, we will not include T4 in our experiment.

### 7.5 Experimental Results

#### 7.5.1 Evaluation: Execution Times for Wired Systems

In the experiments, we observe the following execution times:

1. Process Time, T1; which consists of generating *Ciphertext*, *Ciphercode*, and *Cipherkey*,

2. Transfer Time, TT; a time taken to purely transfer the message to and get a reply from the recipient,

3. Process Time at the Recipient’s side, T3, and

4. Transaction Time, TransacT; a time taken to complete a circular message exchange.
7.5.1.1 Processing Time (T1)

As explained before, T1 is the time taken to generate Ciphertext, Ciphercode, and CipherKey. Table 7-3 and 7-4 shows the result of generating Ciphertext for the MAgeSeM-based and Socket-based systems respectively.

Table 7-3: Time measurements for Ciphertext (MAgSeM-based)

<table>
<thead>
<tr>
<th>Security Mechanism</th>
<th>1M</th>
<th>3M</th>
<th>5M</th>
<th>7M</th>
<th>9M</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 256 + SSL</td>
<td>84.40</td>
<td>221.40</td>
<td>352.13</td>
<td>496.93</td>
<td>631.83</td>
</tr>
<tr>
<td>AES 192</td>
<td>83.80</td>
<td>218.73</td>
<td>339.50</td>
<td>465.63</td>
<td>591.80</td>
</tr>
<tr>
<td>AES 128</td>
<td>77.03</td>
<td>205.23</td>
<td>314.03</td>
<td>429.57</td>
<td>543.77</td>
</tr>
</tbody>
</table>

Table 7-4: Time measurements for Ciphertext (Socket-based)

<table>
<thead>
<tr>
<th>Security Mechanism</th>
<th>1M</th>
<th>3M</th>
<th>5M</th>
<th>7M</th>
<th>9M</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 256 + SSL</td>
<td>82.20</td>
<td>219.20</td>
<td>354.67</td>
<td>485.47</td>
<td>615.20</td>
</tr>
<tr>
<td>AES 192</td>
<td>80.73</td>
<td>208.30</td>
<td>327.10</td>
<td>463.07</td>
<td>583.30</td>
</tr>
<tr>
<td>AES 128</td>
<td>70.80</td>
<td>191.10</td>
<td>301.03</td>
<td>427.73</td>
<td>540.57</td>
</tr>
</tbody>
</table>

From the tables, the encryption time increased with increased plaintext sizes. The results have shown that AES 128-bit has superiority against the AES 192-bit and AES 256-bit in an ascending order. We observed that for AES algorithms, shorter key lengths have faster encryption rate compared to longer key lengths, for all plaintext sizes. The overall time for Socket-based system to produce Ciphertext was slightly faster than the MAgeSeM-based system.
Figure 7-6: Generating Cipherkey and Ciphercode for MAgSeM-based system

Figure 7-7: Generating Cipherkey and Ciphercode for Socket-based system
For *Cipherkey* and *Ciphercode* generation, the results are shown in Figure 7-6 for the MAgSeM-based system and Figure 7-7 for the Socket-based system. The generation of both *Cipherkey* and *Ciphercode* for all of the algorithms took less than 1 millisecond. The generation of *Ciphercode* using $K2$ is slower than the generation of *Cipherkey* using asymmetric key ($pubKr$). We also observed similar performances for both systems. This is because the overall size of data to be encrypted to generate *Cipherkey* ($E(K2, H(Cd)pubKr)$) and *Ciphercode* ($E(T, S, Cd)K2$) did not change for both systems.

For the overall time taken for $T1$, that is, the overall processing time at the sender side, the results for each MAgSeM-based and Socket-based system are shown in Figure 7-8 and Figure 7-9 respectively. Similar pattern of results were generated for both systems, where the processing time increased with increased plaintext sizes.

![Figure 7-8: Comparison for T1 for MAgSeM-based system](image-url)
The MAgSeM-based system took a less time to complete T1 compared to the Socket-based system. Table 7-5 and 7-6 shows the value of T1 for both systems. We calculated the percentage increase/decrease of MAgSeM-based system against the Socket-based system as:

\[
\text{Percentage Increase/Decrease} = \left(\frac{\text{Socket}_\text{value} - \text{MAgSeM}_\text{value}}{\text{MAgSeM}_\text{value}}\right) \times 100
\]

Table 7-5: T1 for MAgSeM-based system

<table>
<thead>
<tr>
<th></th>
<th>1M</th>
<th>3M</th>
<th>5M</th>
<th>7M</th>
<th>9M</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 256+SSL</td>
<td>845.23</td>
<td>1408.80</td>
<td>1866.06</td>
<td>2355.767</td>
<td>3056.33</td>
</tr>
<tr>
<td>AES 192</td>
<td>1000.46</td>
<td>1520.43</td>
<td>1958.30</td>
<td>2417.767</td>
<td>3158.43</td>
</tr>
<tr>
<td>AES 128</td>
<td>875.63</td>
<td>1506.76</td>
<td>1946.33</td>
<td>2347.367</td>
<td>3131.90</td>
</tr>
</tbody>
</table>

Table 7-6: T1 for Socket-based system

<table>
<thead>
<tr>
<th></th>
<th>1M</th>
<th>3M</th>
<th>5M</th>
<th>7M</th>
<th>9M</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 256+SSL</td>
<td>1057.76</td>
<td>1598.40</td>
<td>2137.93</td>
<td>2680.80</td>
<td>3180.76</td>
</tr>
<tr>
<td>AES 192</td>
<td>1558.33</td>
<td>2078.63</td>
<td>2610.96</td>
<td>3304.23</td>
<td>3787.56</td>
</tr>
<tr>
<td>AES 128</td>
<td>1557.80</td>
<td>2065.10</td>
<td>2592.23</td>
<td>3270.80</td>
<td>3742.66</td>
</tr>
</tbody>
</table>
Table 7-7: Percentage increased of the MAgSeM-based system for T1

<table>
<thead>
<tr>
<th></th>
<th>1M</th>
<th>3M</th>
<th>5M</th>
<th>7M</th>
<th>9M</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 256+SSL</td>
<td>25.15</td>
<td>13.46</td>
<td>14.57</td>
<td>13.80</td>
<td>4.07</td>
</tr>
<tr>
<td>AES 192</td>
<td>55.76</td>
<td>36.71</td>
<td>33.33</td>
<td>36.66</td>
<td>19.92</td>
</tr>
<tr>
<td>AES 128</td>
<td>77.91</td>
<td>37.06</td>
<td>33.19</td>
<td>39.34</td>
<td>19.50</td>
</tr>
</tbody>
</table>

As observed in Table 7-7, the MAgSeM-based system performed faster than the Socket-based system, since it used less time to complete T1. For example, for AES 192-bit key with 5M plaintext size, the MAgSeM-based system performed 33.33% faster against the Socket-based system. It could also be noted that for both systems the processing time combining data and channel security (AES 256-bit combined with SSL), gave the fastest processing time.

7.5.1.2  Processing time at the recipient’s side (T3)

T3 started when RA received a request to process a message from MA, until RA gave a signed token to MA.
Figure 7-10 and Figure 7-11 show the results of T3 for the MAgSeM-based and Socket-based systems respectively. From the figures, we observed that for both systems, AES 256-bit performed faster than AES 192-bit and AES 128-bit. We also learned that the MAgSeM-based system performed slower than the Socket-based system to complete T3.

Table 7-10 shows the percentage decreased of the MAgSeM-based system, which were calculated from T3 values of both systems (given in Table 7-8 for MAgSeM-based and Table 7-9 for Socket-based systems). From Table 7-10, we observed that the percentage decreased resulted from AES 192-bit and AES 128-bit gave quite similar output. AES 256-bit performed the slowest among the three algorithms.

<table>
<thead>
<tr>
<th>Plaintext Sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>AES 256+SSL</td>
</tr>
<tr>
<td>AES 192</td>
</tr>
<tr>
<td>AES 128</td>
</tr>
</tbody>
</table>
Table 7-9: T3 for Socket-based system

<table>
<thead>
<tr>
<th></th>
<th>1M</th>
<th>3M</th>
<th>5M</th>
<th>7M</th>
<th>9M</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 256+SSL</td>
<td>1705.83</td>
<td>3785.47</td>
<td>5862.00</td>
<td>9111.03</td>
<td>11917.27</td>
</tr>
<tr>
<td>AES 192</td>
<td>2147.90</td>
<td>4142.23</td>
<td>6288.57</td>
<td>9456.90</td>
<td>12445.87</td>
</tr>
<tr>
<td>AES 128</td>
<td>2146.83</td>
<td>4133.77</td>
<td>6283.90</td>
<td>9405.17</td>
<td>12400.03</td>
</tr>
</tbody>
</table>

Table 7-10: Percentage decreased of the MAgSeM-based system for T3

<table>
<thead>
<tr>
<th></th>
<th>1M</th>
<th>3M</th>
<th>5M</th>
<th>7M</th>
<th>9M</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 256+SSL</td>
<td>-23.17</td>
<td>-35.06</td>
<td>-37.54</td>
<td>-36.65</td>
<td>-53.79</td>
</tr>
<tr>
<td>AES 192</td>
<td>-8.19</td>
<td>-29.95</td>
<td>-33.33</td>
<td>-36.21</td>
<td>-55.46</td>
</tr>
<tr>
<td>AES 128</td>
<td>-8.05</td>
<td>-29.95</td>
<td>-33.25</td>
<td>-36.52</td>
<td>-55.40</td>
</tr>
</tbody>
</table>

7.5.1.3 Transfer Time (TT)

TT represents the time taken only transferring the message to and getting the message from the recipient. The reason for analysing TT is that we would like to observe the time intervals between MAgSeM-based and Socket-based systems, and to know how SSL impacts on the communication overheads.

Figure 7-12: Comparison for TT for MAgSeM-based system
Figure 7-12 and 7-13 compare Transfer Time (TT) for the MAgSeM-based and Socket-based systems respectively. Both systems gave the same pattern of results, where TT was significantly increased when SSL is present. The reason is that, SSL needed more time to encrypt the data during the transmission through the channel connecting the sender and recipient. There was not much difference in transfer time between AES 192-bit and 128-bit.

In general, the MAgSeM-based system has a higher transfer rate compared with the Socket-based system. The reason is that, the migration of mobile agent using FrTP migration strategies (discussed in Section 6.2.1.3) to transfer large data sizes utilized multiple ACL messages, which added to the total transfer time.

The values of TT for both systems are shown in Table 7-11 for the MAgSeM-based system and Table 7-12 for the Socket-based system. We found a decreased of percentage for the MAgSeM-based system (shown in Table 7-13), where all layers performed nearly 100% slower than the Socket-based system.
Table 7-11: TT for MAgSeM-based system

<table>
<thead>
<tr>
<th></th>
<th>1M</th>
<th>3M</th>
<th>5M</th>
<th>7M</th>
<th>9M</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 256+SSL</td>
<td>6084.33</td>
<td>19759.50</td>
<td>26646.47</td>
<td>38298.97</td>
<td>51665.73</td>
</tr>
<tr>
<td>AES 192</td>
<td>5380.30</td>
<td>18496.33</td>
<td>24695.70</td>
<td>35712.93</td>
<td>48562.00</td>
</tr>
<tr>
<td>AES 128</td>
<td>5379.33</td>
<td>18503.27</td>
<td>24692.60</td>
<td>35701.07</td>
<td>48164.03</td>
</tr>
</tbody>
</table>

Table 7-12: TT for Socket-based system

<table>
<thead>
<tr>
<th></th>
<th>1M</th>
<th>3M</th>
<th>5M</th>
<th>7M</th>
<th>9M</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 256+SSL</td>
<td>238.93</td>
<td>630.17</td>
<td>1019.73</td>
<td>1510.90</td>
<td>1830.67</td>
</tr>
<tr>
<td>AES 192</td>
<td>141.57</td>
<td>401.63</td>
<td>638.07</td>
<td>969.10</td>
<td>1199.03</td>
</tr>
<tr>
<td>AES 128</td>
<td>140.73</td>
<td>391.67</td>
<td>646.73</td>
<td>971.97</td>
<td>1186.43</td>
</tr>
</tbody>
</table>

Table 7-13: Percentage decreased of the MAgSeM-based system for TT

<table>
<thead>
<tr>
<th></th>
<th>1M</th>
<th>3M</th>
<th>5M</th>
<th>7M</th>
<th>9M</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 256+SSL</td>
<td>-96.07</td>
<td>-96.81</td>
<td>-96.17</td>
<td>-96.05</td>
<td>-96.46</td>
</tr>
<tr>
<td>AES 128</td>
<td>-97.38</td>
<td>-97.88</td>
<td>-97.38</td>
<td>-97.28</td>
<td>-97.54</td>
</tr>
</tbody>
</table>

7.5.1.4 Transaction Time (TransacT)

The results for TransacT for the MAgSeM-based system and the Socket-based system are shown in Figure 7-14 and Figure 7-15 respectively. We found a different pattern for these results. In the MAgSeM-based system, the communication transactions for Layer 1 that utilized data and channel security (AES 256-bit and SSL) gave the longest execution time. This is in contrast with the Socket-based system, where Layer 1 performed faster than Layer 2 (AES 192-bit) and Layer 3 (AES 128-bit). Communication at Layer 3 was slightly faster than Layer 2 for both systems.
Figure 7-14: Execution time for TransacT for MAgSeM-based

Figure 7-15: Execution time for TransacT for Socket-based
For both systems, the communication that used SSL-only security performed better than the other communications. This is because only the SSL channel was established in the communication, and thus it did not require any process for encrypting, decrypting, hashing, or signing any plaintext. Moreover, the size of the plaintext transferred by the mobile agent did not change.

For the other communications (other than the SSL-only communication), the data security processes added to the total data size that should be carried by the mobile agent, which was almost twice the size of the original plaintext. For example, for a plaintext with the size of 5Mb, the total size of data (containing Ciphertext, Cipherkey, and Ciphercode) carried by MA to the recipient’s platform was about 8.9Mb.

For the overall results of TransacT for both systems, the MAgSeM-based system performed slower compared to the Socket-based system. We calculated the percentage decreased of the MAgSeM-based system in Table 7-16, which were obtained from TransacT values of both systems (as presented in Table 7-14 and 7-15). The decreased of percentages indicated that the MAgSeM-based system took more time to complete TT, which was more than 50% slower compared to the Socket-based system. This is mainly because of the transfer time (TT) caused by JADE.

<table>
<thead>
<tr>
<th>Table 7-14: TransacT for MAgSeM-based system</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>AES 256+SSL</td>
</tr>
<tr>
<td>AES 192</td>
</tr>
<tr>
<td>AES 128</td>
</tr>
<tr>
<td>SSL</td>
</tr>
<tr>
<td>No security</td>
</tr>
</tbody>
</table>
Table 7-15: Transact for Socket-based system

<table>
<thead>
<tr>
<th>Security Setting</th>
<th>1M</th>
<th>3M</th>
<th>5M</th>
<th>7M</th>
<th>9M</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 256+SSL</td>
<td>3002.53</td>
<td>6014.03</td>
<td>9019.67</td>
<td>13302.73</td>
<td>16928.70</td>
</tr>
<tr>
<td>AES 192</td>
<td>3847.80</td>
<td>6614.03</td>
<td>9537.60</td>
<td>13730.23</td>
<td>17386.63</td>
</tr>
<tr>
<td>AES 128</td>
<td>3845.37</td>
<td>6599.00</td>
<td>9522.87</td>
<td>13647.93</td>
<td>17374.97</td>
</tr>
<tr>
<td>SSL</td>
<td>746.80</td>
<td>1170.30</td>
<td>1642.30</td>
<td>2176.80</td>
<td>2535.80</td>
</tr>
<tr>
<td>No security</td>
<td>337.60</td>
<td>714.20</td>
<td>1081.60</td>
<td>1518.60</td>
<td>1873.50</td>
</tr>
</tbody>
</table>

Table 7-16: Percentage decreased of the MAgSeM-based system for TransacT

<table>
<thead>
<tr>
<th>Security Setting</th>
<th>1M</th>
<th>3M</th>
<th>5M</th>
<th>7M</th>
<th>9M</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 256+SSL</td>
<td>-67.19</td>
<td>-74.03</td>
<td>-76.20</td>
<td>-76.26</td>
<td>-78.97</td>
</tr>
<tr>
<td>AES 192</td>
<td>-55.88</td>
<td>-69.92</td>
<td>-73.56</td>
<td>-74.07</td>
<td>-78.17</td>
</tr>
<tr>
<td>AES 128</td>
<td>-55.23</td>
<td>-69.97</td>
<td>-73.60</td>
<td>-74.18</td>
<td>-77.79</td>
</tr>
<tr>
<td>SSL</td>
<td>-73.60</td>
<td>-79.35</td>
<td>-80.02</td>
<td>-79.81</td>
<td>-81.26</td>
</tr>
<tr>
<td>No security</td>
<td>-84.07</td>
<td>-84.98</td>
<td>-85.05</td>
<td>-84.20</td>
<td>-84.54</td>
</tr>
</tbody>
</table>

7.5.1.5 Security Overheads

We define security overhead as the additional time consumed by both systems when security protocols are applied to the communications. For security overhead calculations, we refer to the following Table 7-17 and Table 7-18. The tables summarise the results for TransacT for the MAgSeM-based and Socket-based systems respectively.

Table 7-17: TransacT values in ms for 7 Mb (MAgSeM-based)

<table>
<thead>
<tr>
<th>Security Setting</th>
<th>7M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1: AES 256+SSL</td>
<td>56037.03</td>
</tr>
<tr>
<td>Layer 2: AES 192</td>
<td>52955.70</td>
</tr>
<tr>
<td>Layer 3: AES 128</td>
<td>52863.50</td>
</tr>
<tr>
<td>Layer 4: SSL</td>
<td>10782.78</td>
</tr>
<tr>
<td>Layer 5: No security</td>
<td>9609.40</td>
</tr>
</tbody>
</table>

Table 7-18: TransacT values in ms for 7 Mb (Socket-based)

<table>
<thead>
<tr>
<th>Security Setting</th>
<th>7M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1: AES 256+SSL</td>
<td>13302.73</td>
</tr>
<tr>
<td>Layer 2: AES 192</td>
<td>13730.23</td>
</tr>
<tr>
<td>Layer 3: AES 128</td>
<td>13647.93</td>
</tr>
<tr>
<td>Layer 4: SSL</td>
<td>2176.80</td>
</tr>
<tr>
<td>Layer 5: No security</td>
<td>1518.60</td>
</tr>
</tbody>
</table>
From both tables, the security overheads for each layer were obtained by calculating the additional time consumed by each layer, compared against the “No security” setting. We compared the result of the “No security” communication with 7 Mb of plaintext in both systems. The percentage of security overhead (PSO) were calculated using the following formula:

\[
PSO = \left( \frac{\text{TransacT}_n - \text{TransacT}_{ns}}{\text{TransacT}_{ns}} \right) \times 100,
\]

where \(\text{TransacT}_n\) is the TransacT for Layer-\(n\), and \(\text{TransacT}_{ns}\) is the TransacT for No security communication.

Table 7-19 shows the PSOs for the MAgSeM-based system. The result shows that the SSL-only communication has a lower overhead with only 12.21% compared with the No Security communication. Layer 1, which applied SSL on top of AES 256-bit encryptions, has the highest overhead of 483.15% in comparison with Layer 2 and Layer 3.

<table>
<thead>
<tr>
<th>Security Setting</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1</td>
<td>483.15</td>
</tr>
<tr>
<td>Layer 2</td>
<td>451.08</td>
</tr>
<tr>
<td>Layer 3</td>
<td>450.12</td>
</tr>
<tr>
<td>SSL only</td>
<td>12.21</td>
</tr>
</tbody>
</table>

For the Socket-based system, we also compared the result of the No Security communications with 7 Mb of plaintext, with the other security settings (depicted in Table 7-20). The highest security overhead was 804.14% for Layer 2, followed by Layer 3 (798.72%), and Layer 1 with
data and channel security imposed 775.99% security overhead. The SSL-only communication predictably has the lowest security overhead at 43.34%.

Although the MAgSeM-based system has poor performance for the overall T3, TT, and TransacT, the system in general has lower percentage of overhead, compared with the Socket-based system. The reason why the MAgSeM-based system produces lower overhead percentage is that, when comparing the No Security results for both systems, we found out that the MAgSeM-based system has already taken longer time to complete the TransacT, which was 9609.40 ms (seen from Table 7-17). This is in contrast with the Socket-based system that only took 1518.60 ms (reading from Table 7-18) to complete the TransacT. Therefore, when calculating the overhead, for instance for Layer 1, we found a big difference between Layer 1 and the No security for the Socket-based system, which is quite the opposite of the MAgSeM-based system. We concluded that the security overheads imposed on both systems were the combination of processes of the two systems and the overheads added by the security protocols. The next section discusses the evaluation for the Wireless systems.

7.5.2 Evaluation: Execution Times for Wireless Systems

In the Wireless systems, we once again investigate the same time intervals for T1, T3, TT, and TransacT for MAgSeM-based and Socket-based systems to study the performance for Layer 2, which applies to mobile devices. We compare the three algorithms of 3DES 168-bit, Blowfish 112-bit, and AES 128-bit; and study the performance of each of them.

7.5.2.1 Processing Time (T1)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>600b</th>
<th>100K</th>
<th>150K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES-168</td>
<td>17.67</td>
<td>365.33</td>
<td>545.44</td>
<td>692.44</td>
</tr>
<tr>
<td>Blow-112</td>
<td>3.67</td>
<td>178.78</td>
<td>221.11</td>
<td>326.78</td>
</tr>
<tr>
<td>AES-128</td>
<td>7.44</td>
<td>255.56</td>
<td>323.44</td>
<td>455.67</td>
</tr>
</tbody>
</table>
Table 7-22: Time measurements for Ciphertext (Socket-based)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>600b</th>
<th>100K</th>
<th>150K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES-168</td>
<td>16.67</td>
<td>357.56</td>
<td>555.78</td>
<td>693.45</td>
</tr>
<tr>
<td>Blow-112</td>
<td>3.56</td>
<td>143.11</td>
<td>189.44</td>
<td>243.00</td>
</tr>
<tr>
<td>AES-128</td>
<td>7.22</td>
<td>221.33</td>
<td>293.22</td>
<td>371.67</td>
</tr>
</tbody>
</table>

Table 7-21 shows the time taken for generating Ciphertext for the MAgSeM-based system and Table 7-22 for the Socket-based system. For both systems, all the algorithms gave the same pattern of results, where Blowfish has the superiority against the other algorithms, followed by AES and 3DES.

For Cipherkey and Ciphercode generations in both systems (shown in Figure 7-16 and Figure 7-17), we observed that the Cipherkey generation using asymmetric key took more time in the Wireless system compared with Ciphercode generation. This is in contrast with Wired system, where generating Cipherkey took less time than Ciphercode.
For comparison of the processing time, T1, we refer to Figure 7-18 and 7-19 for MAgsSeM-based and Socket-based systems respectively. For both systems, we noticed that the time taken increased with increased plaintext sizes. In addition, Blowfish performed faster, followed by AES and 3DES.
For the overall comparison, the MAgSeM-based system performed slower than the Socket-based system, which is different in the Wired system. We found less than 20% decrease of the percentage values for all layers in the MAgSeM-based system, as presented in Table 7-25. The percentage values were calculated from the values of T1 for both systems (shown in Table 7-23 and 7-24).

<table>
<thead>
<tr>
<th>Plaintext Sizes</th>
<th>MAgSeM-based system</th>
<th>Socket-based system</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3DES-168</td>
<td>Blowfish-112</td>
</tr>
<tr>
<td>600b</td>
<td>774.67</td>
<td>729.89</td>
</tr>
<tr>
<td>100K</td>
<td>1744.11</td>
<td>1623.56</td>
</tr>
<tr>
<td>150K</td>
<td>2421.11</td>
<td>1727.44</td>
</tr>
<tr>
<td>200K</td>
<td>2826.00</td>
<td>2212.67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Plaintext Sizes</th>
<th>Processing Time (Socket)</th>
</tr>
</thead>
<tbody>
<tr>
<td>600b</td>
<td>3DES-168</td>
</tr>
<tr>
<td>100K</td>
<td>660.33</td>
</tr>
<tr>
<td>150K</td>
<td>1716.67</td>
</tr>
<tr>
<td>200K</td>
<td>2273.78</td>
</tr>
</tbody>
</table>

Figure 7-19: Comparison for T1 for Socket-based
Table 7-25: Percentage decreased of the MAgSeM-based system for T1

<table>
<thead>
<tr>
<th></th>
<th>600b</th>
<th>100K</th>
<th>150K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES 168</td>
<td>-14.76</td>
<td>-1.57</td>
<td>-6.09</td>
<td>-14.08</td>
</tr>
<tr>
<td>Blowfish 112</td>
<td>-9.76</td>
<td>-4.21</td>
<td>-6.05</td>
<td>-10.29</td>
</tr>
<tr>
<td>AES 128</td>
<td>-19.65</td>
<td>-4.32</td>
<td>-2.91</td>
<td>-11.24</td>
</tr>
</tbody>
</table>

7.5.2.2 Processing time at the recipient's side (T3)

Figure 7-20 and Figure 7-21 show the results of T3 for MAgSeM-based and Socket-based systems respectively. From the figures, we found out that the MAgSeM-based system performed faster than the Socket-based system to complete T3. We also observed that for the MAgSeM-based system, the increased plaintext sizes did not give significant difference to T3, in contrast with the Socket-based system.

Figure 7-20: Comparison for T3 for MAgSeM-based
The percentage increased is given in Table 7-28, calculated from T3 values from both systems, shown in Table 7-26 Table 7-27 for the MAgeM-based and Socket-based systems respectively. We observed that the MAgeM-based system performed over 70% faster than the Socket-based system.

**Table 7-26: T3 for MAgeM-based system**

<table>
<thead>
<tr>
<th></th>
<th>600b</th>
<th>100K</th>
<th>150K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES 168</td>
<td>638.78</td>
<td>1019.22</td>
<td>1027.89</td>
<td>1027.67</td>
</tr>
<tr>
<td>Blowfish-112</td>
<td>609.44</td>
<td>980.89</td>
<td>984.22</td>
<td>1052.33</td>
</tr>
<tr>
<td>AES-128</td>
<td>600.89</td>
<td>989.56</td>
<td>979.11</td>
<td>1032.78</td>
</tr>
</tbody>
</table>

**Table 7-27: T3 for Socket-based system**

<table>
<thead>
<tr>
<th></th>
<th>600b</th>
<th>100K</th>
<th>150K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES-168</td>
<td>1138.00</td>
<td>1888.00</td>
<td>2173.44</td>
<td>2586.67</td>
</tr>
<tr>
<td>Blowfish-112</td>
<td>1078.11</td>
<td>1860.22</td>
<td>2119.667</td>
<td>2538.222</td>
</tr>
<tr>
<td>AES-128</td>
<td>1026.00</td>
<td>1854.89</td>
<td>2090.33</td>
<td>2534.78</td>
</tr>
</tbody>
</table>
Table 7-28: Percentage increased of the MAgSeM-based system for T3

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>600b</th>
<th>100K</th>
<th>150K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES-168</td>
<td>78.15</td>
<td>85.24</td>
<td>111.45</td>
<td>151.70</td>
</tr>
<tr>
<td>Blowfish-112</td>
<td>76.90</td>
<td>89.65</td>
<td>115.36</td>
<td>141.20</td>
</tr>
<tr>
<td>AES-128</td>
<td>70.75</td>
<td>87.45</td>
<td>113.49</td>
<td>145.43</td>
</tr>
</tbody>
</table>

7.5.2.3 Transfer Time (TT)

Figure 7-22 shows the comparisons for TT for the MAgSeM-based system. The result shows that Blowfish has the fastest TT, followed by AES and 3DES. The same pattern can be seen in the Socket-based system, shown in Figure 7-23, where 3DES has the slowest transfer rate, while Blowfish performed very well compared with the other two algorithms.
Table 7-29 and Table 7-30 present the values of TT for both systems. For the overall TT comparison, the MAgSeM-based system performed significantly faster than the Socket-based system, where we observed the increased of percentage shown in Table 7-31.

Table 7-29: TT for MAgSeM-based system

<table>
<thead>
<tr>
<th></th>
<th>600b</th>
<th>100K</th>
<th>150K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>3DES 168</strong></td>
<td>171.56</td>
<td>544.33</td>
<td>814.11</td>
<td>1095.22</td>
</tr>
<tr>
<td><strong>Blowfish 112</strong></td>
<td>133.67</td>
<td>508.22</td>
<td>721.00</td>
<td>1059.44</td>
</tr>
<tr>
<td><strong>AES 128</strong></td>
<td>144.44</td>
<td>515.00</td>
<td>781.22</td>
<td>1080.56</td>
</tr>
</tbody>
</table>
Table 7-30: TT for Socket-based system

<table>
<thead>
<tr>
<th></th>
<th>600b</th>
<th>100K</th>
<th>150K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES 168</td>
<td>1362.22</td>
<td>2599.33</td>
<td>3149.89</td>
<td>3571.67</td>
</tr>
<tr>
<td>Blowfish 112</td>
<td>1322.56</td>
<td>2279.67</td>
<td>2252.89</td>
<td>2785.67</td>
</tr>
<tr>
<td>AES 128</td>
<td>1417.33</td>
<td>2434.44</td>
<td>2653.89</td>
<td>3321.78</td>
</tr>
</tbody>
</table>

Table 7-31: Percentage increase of the MAgSeM-based system for TT

<table>
<thead>
<tr>
<th></th>
<th>600b</th>
<th>100K</th>
<th>150K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES 168</td>
<td>694.04</td>
<td>377.53</td>
<td>286.91</td>
<td>226.11</td>
</tr>
<tr>
<td>Blowfish 112</td>
<td>889.44</td>
<td>348.56</td>
<td>212.47</td>
<td>162.94</td>
</tr>
<tr>
<td>AES 128</td>
<td>881.23</td>
<td>372.71</td>
<td>239.71</td>
<td>207.41</td>
</tr>
</tbody>
</table>

The reason is that, the Socket-based system communication was based on the socket objects. In Java, the socket objects in the Wired and Wireless systems have different implementations, although the way they worked is the same. These different implementations required different methods to send and receive messages\(^70\) for different environments. In contrast with JADE agents, all communication between agents were performed using ACL message, regardless of the environment.

\(^70\) As described in Section 6.4.1
7.5.2.4 Transaction Time (TransacT)

For the transaction time for both systems, we refer to Figure 7-24 (MAgSeM-based system) and 7-25 (Socket-based system). The TransacT values for 3DES gave the highest value, while TransacT values produced by Blowfish were the lowest or the fastest, followed by AES. For overall transaction time, the MAgSeM-based system performed faster than the Socket-based
system. We calculated the percentage increased of the MAgSeM-based system from the TransacT values of the two systems, as shown in Table 7-32 and Table 7-33. The results are presented in Table 7-34, which show that the MAgSeM-based system performed over 60% faster than the Socket-based system to complete TransacT.

Table 7-32: TransacT for MAgSeM-based system

<table>
<thead>
<tr>
<th></th>
<th>600b</th>
<th>100K</th>
<th>150K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES 168</td>
<td>1557.89</td>
<td>3280.22</td>
<td>4115.78</td>
<td>4536.33</td>
</tr>
<tr>
<td>Blowfish 112</td>
<td>1473.00</td>
<td>3051.11</td>
<td>3492.89</td>
<td>4360.22</td>
</tr>
<tr>
<td>AES 128</td>
<td>1550.67</td>
<td>3198.67</td>
<td>3576.00</td>
<td>4412.33</td>
</tr>
<tr>
<td>No Security</td>
<td>136.00</td>
<td>1291.67</td>
<td>1669.22</td>
<td>2146.00</td>
</tr>
</tbody>
</table>

Table 7-33: TransacT for Socket-based system

<table>
<thead>
<tr>
<th></th>
<th>600b</th>
<th>100K</th>
<th>150K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES 168</td>
<td>3060.56</td>
<td>6014.78</td>
<td>7744.44</td>
<td>8984.33</td>
</tr>
<tr>
<td>Blowfish 112</td>
<td>3059.33</td>
<td>5757.44</td>
<td>5995.56</td>
<td>7308.89</td>
</tr>
<tr>
<td>AES 128</td>
<td>3068.67</td>
<td>6039.44</td>
<td>6676.33</td>
<td>8470.44</td>
</tr>
<tr>
<td>No Security</td>
<td>928.56</td>
<td>2746.44</td>
<td>3345.33</td>
<td>4285.11</td>
</tr>
</tbody>
</table>

Table 7-34: Percentage increased of the MAgSeM-based system for TransacT

<table>
<thead>
<tr>
<th></th>
<th>600b</th>
<th>100K</th>
<th>150K</th>
<th>200K</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES 168</td>
<td>96.46</td>
<td>83.36</td>
<td>88.16</td>
<td>98.05</td>
</tr>
<tr>
<td>Blowfish 112</td>
<td>107.69</td>
<td>88.70</td>
<td>71.65</td>
<td>67.63</td>
</tr>
<tr>
<td>AES 128</td>
<td>97.89</td>
<td>88.81</td>
<td>86.70</td>
<td>91.97</td>
</tr>
<tr>
<td>No Security</td>
<td>582.76</td>
<td>112.63</td>
<td>100.41</td>
<td>99.68</td>
</tr>
</tbody>
</table>
7.5.2.5 Security Overheads

The overhead calculations for both systems are summarised in the following tables. We compared the result of the TransacT for No security communication with the other layers for 200 Kb of plaintext, shown in Table 7-35 and Table 7-36.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>200 Kb</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES-168</td>
<td>4536.33</td>
</tr>
<tr>
<td>Blowfish-112</td>
<td>4360.22</td>
</tr>
<tr>
<td>AES-128</td>
<td>4412.33</td>
</tr>
<tr>
<td>No Security</td>
<td>2146.00</td>
</tr>
</tbody>
</table>

Table 7-36: TransacT values in ms for 200 Kb (Socket-based)

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>200 Kb</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES-168</td>
<td>8984.33</td>
</tr>
<tr>
<td>Blowfish-112</td>
<td>7308.89</td>
</tr>
<tr>
<td>AES-128</td>
<td>8470.44</td>
</tr>
<tr>
<td>No Security</td>
<td>4285.11</td>
</tr>
</tbody>
</table>

The comparison method used is the same as the one in Section 7.5.1.5. The percentage of security overheads (PSO) for both systems are shown in Table 7-37 for the MAgSeM-based system and Table 7-38 for the Socket-based system.

<table>
<thead>
<tr>
<th>Security setting</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES</td>
<td>111.39</td>
</tr>
<tr>
<td>Blowfish</td>
<td>103.18</td>
</tr>
<tr>
<td>AES</td>
<td>105.61</td>
</tr>
</tbody>
</table>

Table 7-38: PSO for 200 Kb communications (Socket-based)

<table>
<thead>
<tr>
<th>Security Setting</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES-168</td>
<td>109.67</td>
</tr>
<tr>
<td>Blowfish-112</td>
<td>70.56</td>
</tr>
<tr>
<td>AES-128</td>
<td>97.67</td>
</tr>
</tbody>
</table>

From the results, we discovered that MAgSeM-based system has a slightly higher overhead compared with the Socket-based system. The PSO values for Blowfish in both systems gave the lowest PSO, followed by AES and 3DES.
7.6 Discussion and Evaluation

We have presented security performances, in terms of end to end execution times for MAgSeM-based and Socket-based, with both wired and wireless systems. The results from the previous sections lead to the following conclusions:

1. For Wired-based systems:
   a. The Ciphertext generations were faster using shorter key lengths. The Cipherkey generations were faster than the generation of Ciphercode.
   b. The overall processing time at the sender side for Layer 1 were faster when both data and channel security are present. Layer 3 performed faster than Layer 2 because of the shorter key lengths.
   c. For the processing time at the recipient’s side, Layer 1 performed faster than the other layer.
   d. Layer 1 has the longest transfer time because of the channel security overhead. Layer 3 performed slightly faster than Layer 2.
   e. Layer 1 gave the longest transaction time in the MAgSeM-based system compared with Layer 2 and 3. This is in contrast with the Socket-based system, where Layer 1 performed well compared with Layer 2 and 3.
   f. Layer 4 with SSL only has the best performance for the transaction time; however, the security strength of the cipher could not be selected beforehand because it is provided automatically by both JADE and Java SSLServerSocket.
   g. For the overall conclusion, the MAgSeM-based system performed slower than the Socket-based system, mainly because of the migration strategies employed by JADE to transfer large data in several ACL messages. However, the system has the advantages in terms of processing time as well as the security overheads.

2. For Wireless-based systems:
   a. The MAgSeM-based system performed well compared with the Socket based system, in terms of T3, TT and TransacT. We could view this from the design perspective, where in the MAgSeM-based system, the inter-agent communications were performed
using ACL message exchange, regardless of the environment types. On the contrary, the sockets objects have different implementations in different environments, which required different methods to send and receive messages.

b. However, the MAgSeM-based system incurs high cost in terms of the security overhead.

c. Blowfish was a faster algorithm compared to AES (Elminaam et al., 2008) regardless of the key lengths. We found out that it performed very well compared with AES and 3DES for T1, TT, and TransacT. Based on these results from the Wireless-based system, we could see that the selection of the algorithms is an important issue to determine the performance of the communications. However, because MLC is a flexible approach, any algorithm can be chosen, as long it can give better performance to the communications.

The result had shown that the MAgSeM-based Wireless system performed well in terms of T3, TT, and TransacT, compared with the Socket-based Wireless system. This is in contrast with MAgSeM-based Wired system, which did not performed well in terms of T3, TT, and TransacT, compared with the Socket-based Wired system.

However, from the design perspective, we argue that the MAgSeM-based system has an advantage because it is developed using JADE, which conformed to FIPA standard. Communications between agents can be developed in a systematic way by utilizing the specifications provided by the standard such as message format, performatives, as well as the protocols to send messages. This advantage can clearly be seen in the MAgSeM-based Wireless system, which performed well in different environments, while utilizing FIPA-based ACL message.

In addition, the agent’s code can be reused in different environments, for instance, the behaviour of the agent that is used in the Wired system, can be slightly modified and reused in the Wireless system. We could see the evidence when we modified the Wired system’s agent behaviour to adapt with the Wireless system, because JADE did not support mobile agent implementation on wireless device (as discussed in Section 6.3.2).
This is contrary to the socket-based approach, where there is no systematic way to manage processes like message exchange and message handling. This lead to the requirements of additional functionalities to cater for the communication processes in different environments.

Evaluation has shown that the framework is capable of providing the requirement of a secure system. According to Pfleeger (Pfleeger & Pfleeger, 2006), a secure system is a system that is capable of addressing three aspects of confidentiality, integrity, and availability. MAgSeM-based system complies to a secure system by giving users the elements of confidentiality, integrity, and availability like the following:

1. Confidentiality is provided in MAgSeM by using encryptions:
   a. Encrypt plaintext with $K1$:\[ Ciphertext = E(P)K1 \]
   b. Encrypt the agent’s code, signature, and $Token$:\[ Ciphercode = E(Cd, S, T)K2 \]
   c. Encrypt $K2$ that is used to create $Ciphercode$:\[ Cipherkey = E(K2, H(Cd))pubKs \]
   d. SSL channel encryption to secure the communication channel

   We prevent any unauthorized user from stealing the information by encrypting $K1$ and $K2$ in such a way that only the recipient can retrieve them.

2. Availability allows an authorized party to access particular data and service when he/she needs to. MAgSeM is designed so that any request to particular data or services is not denied if it came from an authorized user, and satisfies MAgSeM’s condition. For example, when CLA receives a request to send a message, CLA must agree to respond if (1) the user is an authorized user (which the certificate has been exchanged), and (2) current request it receives not exceed the maximum number allowed at on time. Other example includes when MA makes a request to sign $Token$, RA must agree to respond if the message received is valid and not tampered.

3. Integrity is provided by prohibiting the information to be modified. Or, if the information is in fact modified, we provide a mechanism to detect it. MAgSeM

71 Described in Section 3.1.2
provides hash messages as a mechanism for the recipient to detect any modification on the data sent to him/her, by recalculating a new hash message and comparing it to the one sent by the sender. If both are matched, then the data is not modified and the integrity is verified.

The secure system presented as MAgSeM also conform to requirements of existing standards to protect information privacy such as HIPPA security rules (HIPAA, 1998), (Privacy-Commissioner, 1998, 2001), and (McCallister et al., 2010; Privacy-Act, 1974). According to these standards, when performing message exchanges that include transmitting sensitive medical information over the network, the communication must be protected by ensuring integrity and confidentiality over the message, as well as entity authorization for the users who send and receive the message.

The MAgSeM model provides a rich and efficient framework to design and implement a multi-layered security system, for distributed problem that have similar characteristics to that of e-health communications, as outlined in Section 3.2.1.

7.7 Summary

A security model based on MAS is presented in this chapter. The agents are skilled with knowledge to cater for the security processes to secure online communications. The mobile agent is used as a supporting tool to carry sensitive data. Cryptographic protocols are used to secure data as well as the mobile agent code. Using this security model, a sender can gain control over the plaintext, because a recipient or any other third party does not know the details of the decryption processes. Experiments have been conducted and tested using JADE platform as a proof-of-concept.

The result showed that Layer 1 communication in MAgSeM-based Wired system has the highest transfer time as well as transaction time, due to data and channel security applied in the layer. However, it performs very well in terms of the overall processing time. Layer 3 that
has shorter key lengths, performed slightly faster than Layer 2 in generating Ciphertext and has the fastest transfer time as well as the transaction time.

The results shown from the MAgSeM-based Wireless system, which use data security for Layer 2, we suggest that selection of the algorithms should be made depending on the needs. The SSL-only communication has the lowest overhead, but with the trade-off of security flexibility, in which the strength of the security in the SSL cipher cannot be changed beforehand.
CHAPTER 8   CONCLUSION AND FUTURE WORK

We have assessed the need to have information to be protected when using online communications and message exchanges. This has underlined the importance of having effective security measure to secure the messages. The overall aim of the study presented in this thesis was to investigate how MAS could cater for different security needs for online communications between two nodes. The communications include different devices and different communicating environments.

We began by presenting the motivation for the research, by introducing current issues and challenges in online communications, specifically in the e-health domain. We also highlighted the research questions, which were the basis of our research.

Then, we gave an overview of current security technologies, including the underlying network architectures and standards. We examined the types of security threats to the network. Studies showed that the threats have raised concerns among user about using Internet-based services and online communications, such as in e-health, Internet banking, retailing, and electronic auctions. The users are concerned about giving out their sensitive information online, because they are afraid that such information might be leaked out, misused, or manipulated by an unauthorized party, and their privacy will thus be compromised. In e-health, doctors are not convinced to use unencrypted email applications to exchange sensitive medical information because of the possibility of the information being intercepted by an unauthorized party.

After understanding the threats to the network, we examined the existing security technologies that were used to secure online services and communications from the threats. To understand
how these existing technologies work, we examined each of the technologies and identified the types of security mechanisms, as well as how they were used to secure a communication between two nodes. From the study, we learned that current security technologies cannot cater for different kind of security needs because of the rigid way the security mechanisms are constructed.

The security level or security strength in the current technologies can only be set to one particular value for all communications sessions. As a consequence, the need for a stronger or a weaker security level in different communications cannot be satisfied, without having to reconfigure the whole communication process. In other words, current security technologies did not support automatic and flexible security for different communications.

Next, we introduced our field of interest, the MAS approach. We studied the characteristics of the agents to provide better techniques to enhance the traditional non-agent based systems. The analysis from past studies that we have made, showed that an agent can be specialized with certain skills and instantiated when needed; mobile, which can be migrated to other host; autonomous, that helps automate processes without or with minimal intervention of users; and working cooperatively with one another. These characteristics were needed to cater for security processes in the communications, which is particularly useful for a distributed system with distributed processes.

Then, we highlighted the gaps in current security technologies and emphasized the importance of having flexible security mechanisms, which allowed us to secure different sensitivity levels of information in an organization. We also highlighted our consideration into using the MAS approach in developing our proposed security model.

After identifying and understanding the gap in our research area, we started to go deep into our problem area, which was the e-health domain, and the need to have a secure environment for online communications as our motivating problem. We addressed the problem by first, identifying the users and the types of communications that occurred in e-health. Then we identified the different types of information in e-health and the different levels of sensitivity of
the information. We classified the information into five categories based on ISO17799 standards, which was according to the sensitivity levels. Secondly, from the classification of the information, we then categorized the communications in e-health into five categories (which we call layers later on), so that we could provide appropriate security mechanisms for each layer of the communication. Lastly, we introduced our MLC model based on five layers of communications. MLC provides two types of security mechanisms, which are data and/or channel security for wired and wireless devices, with a range of security strengths provided through the symmetric encryptions.

However, before designing our security framework, we explored in detail the MAS approach and studied the desired agents’ characteristics. We emphasized our justifications for using MAS approach over the traditional non-agent based approach. We discussed the disadvantages of the traditional approach and how MAS enhanced the traditional approach with the desired characteristics to cater for the security processes provided by MLC. We studied in details the desired characteristics and justified how these characteristics can fit into our security model, namely MAgSeM.

Based on our understanding of the MAS approach, we designed MAgSeM, our proposed multi-agent based security model that implemented MLC, which goal was to secure the communications between two nodes (a sender and a recipient) in e-health. We identified each agent’s actions and functionalities that were performed to complete the overall goal of MAgSeM. Then, we organized the agents into layers in such a way that the lower layer agent sent their partial results or reports on finishing their tasks to the higher layer agents, which was part of the cooperation activities among the agents to achieve the overall goal. After that, we explained MAgSeM’s architecture in details, including the types of agents and their respective actions or functionalities. We also explained how the security mechanisms that were specified in MLC were provided for a particular communication, using the default layer and com_layer. We argued that the method of selecting the algorithms presented in MAgSeM is more flexible compared to the method in current technologies. MAgSeM provides different security levels for different types of communications, without having to reconfigure the security setting on the system. This is the opposite of the static implementation of security mechanisms in current
security technologies. In addition, we emphasized that any suitable algorithms can be added to MAgSeM, as long as it complies with the security specification in the MLC Model.

MAgSeM concentrated on giving control on the sender’s side by keeping the information used to recover the plaintext a secret, which is kept at the sender’s side until it is needed. As a consequence, the plaintext sent to recipient cannot be recovered by any unauthorized party. In addition, Token used by the sender, can be a way to check whether the mobile agent has been correctly executed on the recipient’s side, where it has not been denied the use of resources needed at the recipient’s side. Furthermore, the recipient did not have to be burdened with the details of the decryption process of the plaintext, as it would be performed by the agent from the sender’s side. Finally, we ended this chapter by concluding that MAgSeM is not only suitable for e-health, but also for other problems with common characteristics with e-health.

After designing MAgSeM, we developed and implemented the MAgSeM system. First, we introduced the supporting tools used to develop MAgSeM, including the platform for agent developments and the cryptographic library. JADE, which was based on Java Language, was used to develop the agent systems for both wired and wireless devices, and Bouncy Castle cryptographic library was used to supply the cryptographic protocols in MAgSeM. Then, we described the agent classes, which were used to develop the MAgSeM-based system. We used FIPA-ACL for inter-agent communications, which provides a systematic way to develop communication protocols for agents with the readily available message format, performatives, as well as the protocols to send messages.

We also describe the non-agent based implementation, which used the Java Socket programming to establish a connection between a sender and a recipient. However, we identified several weaknesses when implementing the non-agent based system. Firstly, the non-agent based implementation did not have inter-agent communications-like functionality, which allows agents to send partial results to each other. This is important for the agent to make decision on their next task. For example, if there is an undelivered message, the agent can take the initiative to resend the message. This is performed autonomously on behalf of the user without or with minimal user interventions.
However, in the Socket-based system, the user is in charge and needs to interact and drive the whole communication process. If there is an undelivered message, the user is responsible for resending the message. In other words, the security processes cannot be fully automated.

Secondly, instead of using agent to perform the decryption processes at the recipient’s side, the Java .class object was used. As is known, the method in the Java objects must be invoked in order to perform certain processes. Therefore, the recipient must know all the names of the methods and classes beforehand, in order to let the object perform the assigned task, which clearly burdened the recipient. Unlike object, an agent has its own data and states and can autonomously perform its action without interventions.

Lastly, as we used FIPA-ACL message format, some of the codes that we used for agent communications and behaviour can be reused and modified slightly for different environments. This is in contrast with the Socket-based system, where we cannot entirely reuse the programs or codes used in wired machine on the wireless device, because of the different specifications and protocols of establishing the connection and sending messages.

After designing and implementing both the systems, we presented our experiments, which goal was to evaluate the performance of communications for each layer. The results and analysis were discussed that compared the execution times in MAgSeM-based and Socket-based systems, including executions in Wired and Wireless systems. From the results, we evaluated how MAgSeM-based system performed in terms of communication performance and how the system managed the security processes, in comparison with Socket-based system. The results of the experiments for MAgSeM-based Wired system showed that Layer 1 that utilized data and channel security (using SSL) incurred the highest communication time compared with Layer 2 and Layer 3 that used only data security. The results demonstrated that SSL added a significant cost to the transfer time.

From the results, we also observed that shorter key lengths gave faster processing time than longer key lengths. However, longer key lengths gave better security protection. The selection of algorithms for every layer is also an important issue to get an optimise performance of
communications. For overall transaction time, MAgSeM-based system performed slower than Socket-based system because of the migration strategy implemented in FrTP to transfer larger data in several ACL messages.

For the Wireless system, MAgSeM-based system performed well compared with Socket-based system. The reason is that MAgSeM-based system communications between agents were performed using ACL message that provides a systematic message format, performatives, as well as protocols to send messages, regardless of the environment types. On the other hand, the implementation of socket objects is different in different environments, which required different methods to send and receive messages.

The developed MAgSeM-based system presents many features that benefit users. It gives a much better control on security to the initiator of the communication with assuring security of the channel and at the recipient’s node. The layered structure improves efficiency based on the level of security decided at different levels. There can be a significant gain of efficiency for the processing and transmission times with a careful selection based on needs of the appropriate layers in the security model. It is worth noting the following points:

1. Flexible security mechanisms: MAgSeM provides a flexible way to choose security strengths for a communication using the MLC model, without having to reconfigure the whole system, which happens in the existing security systems.

2. Security automation: the use of MAS gives the user an automation feature to the system. This is the result of the autonomous characteristic of the agent that allowed agent to perform tasks with minimal or without intervention from the user.

3. The use of FIPA-ACL gives advantages in terms of design and implementation-wise, especially for systems that involve heterogeneous environments. The implementation of ACL message to exchange messages between two nodes is the same regardless of the environment. Thus, the user only needs to slightly modify the behaviours of the agents to cater for the desired security processes, without worrying about additional programming for sending and receiving messages on different environments.
8.1 Assessment against Research Questions

In this section, we revisit the research questions presented in Chapter 1, and examine whether we have attained the appropriate answer to the questions.

1. *What are the standard criteria or the requirements for a secure system?*
   According to Pfleeger (Pfleeger & Pfleeger, 2006), a secure system is a system that is capable of addressing three aspects of: confidentiality, integrity, and availability. MAgSeM-based system complies with the above requirements, as discussed in Section 7.5.

2. *How can we handle the different security needs for different types of communications?*
   The different security needs emerged from the rigid way of how the current security technologies provide security mechanisms. Current practices do not satisfy the requirements for flexible security strengths. Users cannot choose their own security preferences for different types of communications transmitting different level of sensitivities. Thus, the different security needs can be handled and catered for if the setup process before the communication starts permits multiple security options that suit the needs. However, the setup process should be handled automatically as not to burden the user.

3. *Can the multi-agent design framework be developed to model security needs and to design a distributed system realising the security requirements?*
   MLC was proposed to model the security needs. It provided flexible security mechanisms through five layers of communications, which represent five types of security, which include data and/or channel security (explained in Section 3.2.2). MAgSeM was designed in such a way that it supported distributed communications, which implemented the MLC model (explained in Chapter 5).

4. *What are the criteria of the multi-agent system approach that can be used to cater for the security processes?*

72 Described in Section 3.1.2
The MAS approach was used in this research to cater for the security processes. We are interested in the MAS characteristics such as the ability to coordinate and cooperate to achieve the overall system goal, autonomous, interactive, extensible, and mobile. The agents in MAgSeM cooperate and coordinate with each other using the organizational structure method, described in Section 4.2.1. In addition, Agents interact with each other for communicating their partial results and reporting their finished tasks, which is important for other agents to complete next tasks. For example, a sender’s agent collaborates with the agent from the recipient’s side in order to get permission to send a message and to determine what kind of algorithm should be in place for the subsequent communication session. Furthermore, MAgSeM can launch a mobile agent to carry a secure message to the recipient’s side and let the agent handles the security processes there, including instantiating a new agent for decryption purposes, without having the user to interfere with the processes (such as described in Section 5.3).

5. How does MAS handle the security mechanisms for the different communications?
MAgSeM uses the extensible property (Section 4.2.3) of the agent to cater for the different communication, which means, an agent can be instantiated and skilled to perform certain tasks. Thus, to handle the different types of communications, an agent is instantiated and then skilled or assigned with appropriate security mechanisms such as specified in MLC, to cater for that particular communication.

6. How does MAS give a much better control on security compared to the traditional non-agent based system?
As described in Section 5.3.3, MAgSeM provides a better control on security by using the mobility, extensibility, and autonomous characteristics of the agents to automate and cater for the security processes. MAgSeM-based system provides a flexible security setting to different types of communications that carry different types of sensitivities of the information. MAgSeM accomplished this by using com_layer, which determines the security mechanisms stored in the MLC specifications. In contrast with the existing non-agent based system, (discussed in Section 2.6), the technologies only provides one type of security mechanism for every communication. Moreover, the security settings of the non-
agent based system need to be reconfigured, in order to change the security level. MAgSeM-based system can automate the security processes in such a way that users do not need reconfigurations.

In MAgSeM, a mobile agent (MA) can be instantiated and migrated to carry a secure message to the recipient side to perform the tasks without user’s intervention. Once arrived at the recipient’s side, RA verifies that MA does really come from the sender’s side. Then, MA is allowed to contact the sender’s side using a token. An advantage of using the token is that, the sender knows that MA has been correctly executed at the recipient’s host and the access to the recipient’s resources is not denied, which is the recipient’s private key. The sender can check that the signature of the token really belongs to the recipient, by verifying it with the recipient’s public key.

An agent can also be instantiated and assigned to hold information for a communication in progress, and dispose the information once the communication has ended. In this case, the agent (SUA) can hold a symmetric key ($K1$) and the information about the key that encrypts plaintext, as well as the disposable asymmetric key ($Ks$) that will be used to produce $hashKey$ later on, until MA asks for the information. By keeping $K1$ a secret at the sender’s side, the sender has the advantage of gaining control over the data that is carried by the mobile agent, where the recipient or any other third party does not need to know about $K1$. Even if an attacker could get a hold of $hashKey$, he/she still cannot recover $K1$, because $Kp$ that is used to decrypt $hashKey$ is at the recipient’s side.

The recipient also has the advantage of not being burdened with the details of decryption processes to recover the plaintext. It will be autonomously done by DA, once it is executed. The recipient only needs to verify that the code of DA indeed came from the sender by checking $S$ and $H(Cd)$. In addition, the recipient can check the integrity of the plaintext by calculating a new hash code from the recovered plaintext and compare it with the one received from sender.
In addition, MAgSeM conforms to the requirements of existing health standards to protect information privacy such as HIPPA security rules (HIPAA, 1998), (Privacy-Commissioner, 1998, 2001), and (McCallister et al., 2010; Privacy-Act, 1974), which requires protection over communication that exchange sensitive medical information.

### 8.2 Future Work

The thesis investigated and has developed a new model for providing flexible security mechanisms to secure online communications, which is based on MAS approach. However, there are some unanswered research questions that need to be addressed, which we identify below:

1. **Support messages with formats other than text messages**
   In our research, we only use plaintext for information exchange. In real practice in e-health, medical information includes medical images, graphics or illustration, video clips, and vector data such as graphs and tables. We could incorporate standards to store and transmit medical data such as described in HL7 ([http://www.hl7.org/](http://www.hl7.org/)) and DICOM ([http://dicom.nema.org/](http://dicom.nema.org/)).

2. **MAgSeM as a Wrapper Application to provide Communication Security**
   At present, MAgSeM is used to secure plaintext that is entered by users through a user interface (refer to Figure 6-10). However, there are many types of applications in e-health with different types of data, such as video conferencing, web-based application such as in Pagliari et al. (2005) and Tang et al. (2003), email technology and other application-specific for delivering services related to healthcare such as in (Bobadilla et al., 2007; Kim et al., 2001; Nguyen et al., 2008). We can consider of MAgSeM, as a wrapper application to be used with these applications. As a result, the applications can be guaranteed of security protection during communications or message exchanges with other parties. To accomplish this, we have to consider the integration with MAgSeM complies with available standards and specifications in a software engineering point of view.
3. Service Oriented Architecture

In our present MAgSeM system, we have highlighted it to be used in other e-service applications. An interesting feature to be investigated is for the security to be published, so that client can request for a security service, and the security will be plugged in as a service. We could accomplish this objective by using service oriented architecture (SOA) to publish security services offered by MAgSeM, and users who are interested to use the service can subscribe to the service.

4. Integration with other security systems

MAgSeM system can be integrated with other security system. For example, MAITS (Sharma et al., 2007a) is a multi-agent based security system that has specialized agents to perform security tasks to secure computer systems, such as administration assistant agents to gather security data, authentication and authorization agents to control access to the system, system log monitoring agents and intrusion detection agents to detect malicious activities, and pre-mortem based computer forensic agents to collect evidence of a particular attacks. MAgSeM can be incorporated in this type of system to help secure communications in two nodes, with flexible security mechanisms.
**Table A1: Function `splitString`**

*Function: To split string `s` into `nom`, and keep the split words in an array `result[]`*

```java
public String[] splitString(String s, String delimiter, int nom){
    String[] result = null;

    if (nom == 1){
        result= new String[2];
        String[] temp1 =null;
        temp1 = s.split(delimiter);
        int index1 = s.indexOf(delimiter);
        String word1 = s.substring(0,index1);
        result[0] = word1;
        String word2 = s.substring(index1+1);
        result[1] = word2;
    } else if (nom == 2){
        result= new String[3];
        String[] temp1 =null;
        temp1 = s.split(delimiter);
        int index1 = s.indexOf(delimiter);
        String word1 = s.substring(0,index1);
        result[0] = word1;
        String temp = s.substring(index1+1);
        String[] temp2 =null;
        temp2 = temp.split(delimiter);
        int index2 = temp.indexOf(delimiter);
        String word2 = temp.substring(0,index2);
        result[1] = word2;
        String word3 = temp.substring(index2+1);
        result[2] = word3;
    } else if (nom == 3){
        result= new String[4];
        String[] temp1 =null;
        temp1 = s.split(delimiter);
        int index1 = s.indexOf(delimiter);
        String word1 = s.substring(0,index1);
        result[0] = word1;
        String temp = s.substring(index1+1);
        String[] temp2 =null;
        temp2 = temp.split(delimiter);
        int index2 = temp.indexOf(delimiter);
        String word2 = temp.substring(0,index2);
        result[1] = word2;
        String temp3 = temp.substring(index2+1);
        String[] temp4 =null;
        temp4 = temp3.split(delimiter);
        int index3 = temp3.indexOf(delimiter);
        String word3 = temp3.substring(0,index3);
        result[2] = word3;
        String word4 = temp3.substring(index3+1);
        result[3] = word4;
    }
    return result;
}
```

**Appendix A: Program Construct for Concepts Developed**
temp4 = temp3.split(delimiter);
int index3 = temp3.indexOf(delimiter);
String word3 = temp3.substring(0,index3);
result[2] = word3;
String word4 = temp3.substring(index3+1);
result[3] = word4;
}
else{
    result = s.split(delimiter);
}
Table A2: Function encrypt()

Function: Symmetric encryption for Wired System

```java
public String encrypt(File in, File out, int choice) throws IOException, InvalidKeyException {
    SecureRandom random = new SecureRandom();
    IvParameterSpec ivSpec = createIvForAES(1, random);

    String SivBytes = new String(Base64.encodeBase64(ivBytes));
    CipherOutputStream os = null;
    FileInputStream is = new FileInputStream(in);
    String s = null;

    if (choice == 1) {// encrypt using K1
        try {
            aesCipher.init(Cipher.ENCRYPT_MODE, keyAes, ivSpec);
        } catch (InvalidAlgorithmParameterException e) {System.out.println(e);}

        os = new CipherOutputStream(new FileOutputStream(out), aesCipher);
        copy(is, os);
        os.close();
    }
    else { // encrypt using K2
        try {
            aesCipher2.init(Cipher.ENCRYPT_MODE, keyAes2, ivSpec);
        } catch (InvalidAlgorithmParameterException e) {System.out.println(e);}

        os = new CipherOutputStream(new FileOutputStream(out), aesCipher2);
        copy(is, os);
        os.close();
    }

    byte[] ciphertext = new byte[(int) out.length()];
    try {
        FileInputStream fileInputStream = new FileInputStream(out);
        fileInputStream.read(ciphertext);
    } catch (Exception e) {System.out.println("File Not Found.");}

    s = new String(Base64.encodeBase64(ciphertext));
    return s;
}
```

// this function is based on (Hook, 2005)’s book Beginning Cryptography in Java
// to create random IV for the key

```java
public IvParameterSpec createIvForAES(int messageNumber, SecureRandom random) {
    byte[] ivBytes = new byte[16];

    // initially randomize
    random.nextBytes(ivBytes);

    // set the message number bytes
    ivBytes[0] = (byte) (messageNumber >> 24);
    ivBytes[1] = (byte) (messageNumber >> 16);
    ivBytes[2] = (byte) (messageNumber >> 8);
```
ivBytes[3] = (byte)(messageNumber >> 0);

// set the counter bytes to 1
for (int i = 0; i != 7; i++){
    ivBytes[8 + i] = 0;
}
ivBytes[15] = 1;

return new IvParameterSpec(ivBytes);
**Table A3: Function saveKey()**

**Function:** Asymmetric encryptions for Wired System, encrypt with a private or a public key

```java
public String saveKey(File out, String info, int choice) throws IOException, GeneralSecurityException {
    FileInputStream fileInputStream = null;
    byte[] str = null;
    String s = null;

    if (choice == 1) { // create hashKey
        String aesKeyStr = new String(Base64.encodeBase64(aesKey));
        // where info = mlc + H(P)
        String aesKeyHashStr = aesKeyStr + "|" + info;
        byte[] strByte = Base64.encodeBase64(aesKeyHashStr.getBytes());
        pkCipher.init(Cipher.ENCRYPT_MODE, privk); // privk = Ks
        CipherOutputStream os = new CipherOutputStream(new FileOutputStream(out), pkCipher);
        os.write(strByte);
        os.close();
    } // end if

    else { // create cipherkey
        String aesKeyStr2 = new String(Base64.encodeBase64(aesKey2));
        // where info = ivBytes for K2 + H(Cd)
        String aesKeyHashStr2 = aesKeyStr2 + "|" + info;
        byte[] strByte2 = Base64.encodeBase64(aesKeyHashStr2.getBytes());
        pkCipher1.init(Cipher.ENCRYPT_MODE, pubK); // pubK = pubKr
        CipherOutputStream os = new CipherOutputStream(new FileOutputStream(out), pkCipher1);
        os.write(strByte2);
        os.close();
    } // end else

    try {
        fileInputStream = new FileInputStream(out);
        str = new byte[(int) out.length()];
        fileInputStream.read(str);
    } catch (Exception e) { System.out.println("File Not Found."); }

    s = new String(Base64.encodeBase64(str));
    return s;
}
```
### Function: Symmetric and Asymmetric Encryptions for Wireless system

//symmetric encryption

```java
public byte[] Encrypting(byte[] data , int choice) throws Exception {
    int size = 0, len = 0; byte[] result=null;

    if (choice ==1){//encrypt with K1
        cipher.init(true, key);
        size = cipher.getOutputSize(data.length);
        result = new byte[size];
        len = cipher.processBytes(data,0,data.length,result,0);

        if(data == null || data.length == 0)
            return new byte[0];
        //process the last block in the buffer
        len += cipher.doFinal(result,len);

        if(len < size){
            byte[] tmp = new byte[len];
            System.arraycopy(result,0,tmp,0,len);
            result = tmp;
        }
    }
    else{//encrypt with K2
        cipher2.init(true, key2);
        size = cipher2.getOutputSize(data.length);
        result = new byte[size];
        len = cipher2.processBytes(data,0,data.length,result,0);

        if(data == null || data.length == 0)
            return new byte[0];
        len += cipher2.doFinal(result,len);

        if(len < size){
            byte[] tmp = new byte[len];
            System.arraycopy(result,0,tmp,0,len);
            result = tmp;
        }
    }
    return result;
}
```

//Asymmetric encryption

```java
public byte[] RSAEncrypt (byte[] toEncrypt, RSAKeyParameters pubkey) throws Exception {
    if (pubkey == null)
        throw new Exception("Generate RSA keys first!");

    AsymmetricBlockCipher eng = new RSAEngine();
    eng = new PKCS1Encoding(eng);
    eng.init(true, pubkey);
    return eng.processBlock(toEncrypt, 0, toEncrypt.length);
}
```
### Agent Classes (Wired System)

**Table A5: MTA Class**

**Function:** excerpts of MTA’s behaviours

```java
public class mtAgent extends Agent{
    ...
    protected void setup()
    {
        addBehaviour(new newMessage());
        addBehaviour(new newRequestAnswer(this));
        addBehaviour(new resultEncrypting(this));
    }
}

class newMessage extends CyclicBehaviour {
    ...
    public void action()
    {
        //wait for a new message from DOA
        mt = MessageTemplate.and(MessageTemplate.MatchConversationId("New-Message"),
                                 MessageTemplate.MatchPerformative(ACLMessage.INFORM));

        ACLMessage msg = blockingReceive(mt);
        FileRead fr = new FileRead();
        fr.loadRL();//read and load the recipient’s list

        if (msg!=null)
        {
            String content = msg.getContent();
            splitMe sp = new splitMe();
            String[] no_rec = sp.splitString(content, |",1);
            num = Integer.parseInt(no_rec[0]); //number of recipients
            recipients = new String[num];
            recipients = sp.splitString(no_rec[1], ";", 4);

            com_layer_array = new int[num];
            address_array = new String[num];

            //for every recipient send MTA a request
            for (int i = 0; i<num; i++)
            {
                String address = fr.getAddress(recipients[i]);
                address_array[i] = address;

                AID aid = new AID();
                aid.setName("CLA@"+address+":1099/JADE");
                aid.addAddresses("http://"+address+":7778/acc");

                ACLMessage msg1 = new ACLMessage(ACLMessage.REQUEST);
                msg1.setConversationId("Send-MSG");
                msg1.addReceiver(aid);
                send(msg1);
            }
        }
    }
}
```
class FileRead{
    private int com_layer;
    private Hashtable determineComLayer = new Hashtable();
    private Hashtable determineAddress = new Hashtable();
    private Hashtable determineRecipient = new Hashtable();

    public void loadRL(){
        try{
            FileInputStream fstream = new FileInputStream("Recipient.txt");
            DataInputStream in = new DataInputStream(fstream);
            BufferedReader br = new BufferedReader(new InputStreamReader(in));

            String strLine;
            //Read File Line By Line
            while ((strLine = br.readLine()) != null) {
                splitMe sp = new splitMe();
                String[] result = sp.splitString(strLine, ",", 3);
                determineComLayer.put(result[0], result[2]);
                determineRecipient.put(result[3], result[0]);
                determineAddress.put(result[0], result[3]);
            } //while
            in.close();
        } catch (Exception e){System.err.println("Error: " + e.getMessage());}
    } //void main

    public int calculateCM(int s, String r){
        String dfFor_r = (String)determineComLayer.get(r);
        int rInt = Integer.parseInt(dfFor_r);

        if(s == rInt)
            com_layer = rInt;
        else if (s < rInt)
            com_layer = rInt;
        else
            com_layer = s;

        return com_layer;
    }

    public String getAddress(String r){
        String add = (String)determineAddress.get(r);
        return add;
    }

    public String getRecipient(String r){
        String recip = (String)determineRecipient.get(r);
        return recip;
    }
    ....
}
class newRequestAnswer extends CyclicBehaviour {
...
    public newRequestAnswer(Agent _a) {...}

    public void action(){
        mt = MessageTemplate.and(MessageTemplate.MatchConversationId("Send-MSG"),
                                  MessageTemplate.MatchPerformative( ACLMessage.REQUEST ));

        ACLMessage msg= blockingReceive(mt);
        if(!(msg.getContent().equals("null"))){
            splitMe sm = new splitMe();
            String[] result1 = sm.splitString(msg.getContent(),"|",2);
            //result1[0]:AGREE/REJECT | result1[1]: RA's name | result1[2]: mlc
            mlcR = result1[2];
            if(result1[0].equals("AGREE")){
                //get the recipient's address
                String[] result = sm.splitString(msg.getSender().getName(),":",1);
                String[] address = sm.splitString(result[0],"@",1);
                String testname;
                do{//check if the name has been used
                    testname = "SUA"+listsua;
                    listsua++;
                }while(!(ls.indexOf(testname)<0));
                ls.add(testname);
                String givetocrypto = address[1]+"|"+testname+"|"+result1[1]+"|"+mlcR;

                //before sending, copy first
                logAddress.put(address[1],givetocrypto);

                ACLMessage msg1 = new ACLMessage(ACLMessage.INFORM);
                msg1.addReceiver (new AID("cryptoAgent", AID.ISLOCALNAME));
                msg1.setConversationId("New-address");
                msg1.setContent(givetocrypto);
                myAgent.send(msg1);
            }
        }
    }
//end class newRequest

class resultEncrypting extends CyclicBehaviour {
...
    public resultEncrypting(Agent _a) {...}

    public void action(){
        mt = MessageTemplate.and(MessageTemplate.MatchConversationId("Result-
                                  Encrypting"),MessageTemplate.MatchPerformative( ACLMessage.INFORM ));
        ACLMessage msg= blockingReceive(mt);
        if(msg!=null){
            splitMe sm = new splitMe();
            String[] ans = sm.splitString(msg.getContent(),"|",1);

            ACLMessage msgs = new ACLMessage(ACLMessage.INFORM);
            msgs.addReceiver (new AID("IA", AID.ISLOCALNAME));
            msgs.setConversationId("New-address");
            msgs.setContent(givetocrypto);
            myAgent.send(msgs);
        }
    }
//end class resultEncrypting

if(ans[0].equals("Finish")){
    int locationIndex = ls.indexOf(ans[1]);
    ls.remove(locationIndex); //remove delivered address from the list
    logAddress.remove(ans[1]);

    msgs.setContent("Success");
    myAgent.send(msgs);
}
else{//ans[1]: address that cannot be delivered
    if (MASTER_COUNT<6){//total count of re-sending message less than 5
        myAgent.doWait(RETRY_TIME);
        AID aid = new AID();
        aid.setName("CLA"+ans[1]+":1099/JADE");
        aid.addAddresses("http://"+ans[1]+":7778/acc");

        ACLMessage msg1 = new ACLMessage(ACLMessage.REQUEST);
        msg1.setConversationId("Send-MSG1");
        msg1.addReceiver( aid);
        send(msg1);
        MessageTemplate mt = MessageTemplate.and
            (MessageTemplate.MatchConversationId("Send-MSG1"),
                MessageTemplate.MatchPerformative(ACLMessage.REQUEST));

        ACLMessage msg2= blockingReceive(mt);

        if(!(msg2.getContent().equals("null"))){
            splitMe sm1 = new splitMe();
            String[] result1 = sm1.splitString(msg2.getContent(),"|",2);

            if(result1[0].equals("AGREE")){
                //get the recipient's address
                String[] result = sm1.splitString(msg2.getSender().getName(),":",1);
                String[] address = sm1.splitString(result[0],"@",1);
                String testname;
                do{
                    testname = "SUA"+listsua;
                    listsua++;
                }while(!(ls.indexOf(testname)<0));
                ls.add(testname);
                String givetocrypto=
                    address[1]+"|"+testname+"|"+result1[1]+"|"+mlcR;
                //result[1]= RA's name

                ACLMessage msg11 = new ACLMessage(ACLMessage.INFORM);
                msg11.addReceiver (new AID("cryptoAgent", AID.ISLOCALNAME));
                msg11.setConversationId("Add-Not-Delivered");
                msg11.setContent(givetocrypto);
                myAgent.send(msg11);

                msgs.setContent("Fail");
                myAgent.send(msgs);
            }
        }
    }
}
```java
Table A6: cryptoAgent class

Function: excerpts of cA’s behaviours

```
```java
public class cryptoAgent extends Agent {
    ...
    protected void setup() {
        addBehaviour(new newAddress(this));
        addBehaviour(new caBehaviour(this));
        addBehaviour(new reSend(this));
    }

    class newAddress extends CyclicBehaviour {
        ...
        public newAddress(Agent _a) {

            public void action() {
                mt = MessageTemplate.and(MessageTemplate.MatchConversationId("New-address"),
                                            MessageTemplate.MatchPerformative(ACLMessage.INFORM));
                ACLMessage ans = blockingReceive();
                if (ans != null) {
                    splitMe sm = new splitMe();
                    String[] nameAdd = sm.splitString(ans.getContent(), "\|", 3);
                    mlcR = nameAdd[3];
                    FileRead fr = new FileRead();
                    fr.loadRL();
                    recipient = fr.getRecipient(nameAdd[0]);
                    int comlayer = fr.calculateCM(dlSender, recipient);
                    MLC m = new MLC(comlayer);
                    String info = m.getMLC();
                    Object[] args1 = new Object[7];
                    args1[0] = info; //mlc for K1
                    args1[1] = nameAdd[0]; //address
                    args1[2] = keystore;
                    args1[3] = truststore;
                    args1[4] = recipient;
                    args1[5] = nameAdd[2]; //RA's name
                    args1[6] = mlcR; //mlc from recipient, for K2
                    String name = nameAdd[1];
                    File out = new File(nameAdd[1] + "Plaintext.txt");
                    try{
```
copy(in, out);
}catch (IOException e){}
AgentContainer c = getContainerController();
try {
    AgentController ad = c.createNewAgent( name, "SetUpAgent", args1);
    ad.start();
}catch (Exception e){}
//if
}//action

public void copy(File src, File dst) throws IOException {
    InputStream in = new FileInputStream(src);
    OutputStream out = new FileOutputStream(dst);

    // Transfer bytes from in to out
    byte[] buf = new byte[1024];
    int len;
    while ((len = in.read(buf)) > 0) {
        out.write(buf, 0, len);
    }
    in.close();
    out.close();
}
}//class

// wait for SUAs's result, if finish, then eleminate from the list (the list that
// store all SUAs) when is it empty, all sua's have returned results, terminate...

class caBehaviour extends CyclicBehaviour{
...
public caBehaviour(Agent _a) {
    super(_a);
    a =_a;
}

public void action(){
    mt = MessageTemplate.and(MessageTemplate.MatchConversationId("Finish-
Encrypting"),
MessageTemplate.MatchPerformative( ACLMessage.INFORM ));
ACLMessage msg = blockingReceive();
if (msg!=null) {
    String answer = msg.getContent();
    splitMe sm = new splitMe();
    String result[] = sm.splitString(answer, "|", 2);

    ACLMessage ans = new ACLMessage(ACLMessage.INFORM);
    ans.addReceiver (new AID("MTA", AID.ISLOCALNAME));
    ans.setConversationId("Result-Encrypting");
    if (result[0].equals("Finish")){
        System.out.println ("\nAgent "+result[2]+" has finished its task..");
        ans.setContent("Finish|"+result[2]);
    }else{
        System.out.println("Cannot Deliver The Message to the following

class reSend extends CyclicBehaviour{
    ...
    public reSend(Agent _a) {...}

    public void action(){
        mt = MessageTemplate.and
            (MessageTemplate.MatchConversationId("Add-Not-Delivered"),
            MessageTemplate.MatchPerformative( ACLMessage.INFORM ));

        ACLMessage ans = blockingReceive();
        if (ans!=null) {
            splitMe sm = new splitMe();
            String[] nameAdd = sm.splitString(ans.getContent(),"|",3);
            mlcR = nameAdd[3];
            FileRead fr = new FileRead();
            fr.loadRL();
            recipient = fr.getRecipient(nameAdd[0]);
            int comlayer = fr.calculateCM(dlSender, recipient);

            MLC m = new MLC(comlayer);
            String info = m.getMLC();
            Object[] args1 = new Object[7];
            args1[0] = info;
            args1[1] = nameAdd[0];//address
            args1[2] = keystore;
            args1[3] = truststore;
            args1[4] = recipient;
            args1[5] = nameAdd[2];//RA's name
            args1[6] = mlcR;

            String name = nameAdd[1] ;
            File out = new File (nameAdd[1]+"Plaintext.txt");
            try{
                copy(in, out);
            }catch(IOException e){}
            AgentContainer c = getContainerController();
            try {
                AgentController ad = c.createNewAgent( name, "SetUpAgent", args1);
                ad.start();
            }catch (Exception e){}
        } //if
    }//action

    ...
}//class

}//class cryptoAgent...
Table A7: SUA class

<table>
<thead>
<tr>
<th>Function: excerpts of SUA’s behaviours</th>
</tr>
</thead>
</table>

public class SetUpAgent extends Agent{

protected void setup(){
    Object[] args = getArguments();

    mlcList = (String) args[0]; //mlc for K1
    address = (String) args[1];
    keystore = (String) args[2];
    truststore = (String) args[3];
    recipient = (String) args[4];
    RAi = (String) args[5];
    mlcR = (String) args[6]; //mlc for K2

    agentName = getLocalName();

    String agentAddress = getName();
    splitMe sm = new splitMe();
    resultAdd = sm.splitString(agentAddress, "@", 1);
    currentAdd = sm.splitString(resultAdd[1], ":", 1);

    try{
        Runtime.getRuntime().exec("jar cf "+agentName+"decryptAgent.jar decryptAgent.class decryptAgent$validCodeEdit.class");
    }catch(java.io.IOException e){}

    ic = new infoController(); //holds (Kp,Ks)

    try{
        dp = new dataPreparation(mlcList, mlcR);
        dp.prepareData(ic, keystore, truststore, recipient, agentName);
        ivbytes = dp.getIVbytes(); //for K1
    }catch (Exception e){System.out.println(e);}

    addBehaviour( new myBehaviour( this));
}

//end setup()

class myBehaviour extends SimpleBehaviour {

public myBehaviour(Agent _a) { ... }

public void action(){
    switch(state){
    case 0:
        String suaName= getName();

        Object [] args = new Object[6];
        args[0] = ic;
        args[1] = agentName;

        ...}
`args[2] = address;`  
`args[3] = suaName;`  
`args[4] = RAi;`  
`args[5] = ivbytes;`  

String name = "test";  
AgentContainer c = getContainerController();  
try {  
    AgentController ad = c.createNewAgent( name, "MobileAgent", args );  
    ad.start();  
} catch (Exception e) {}  
state++;  
break;  

```java
state++;
break;
```

case 1:
    mt = MessageTemplate.and(MessageTemplate.MatchConversationId("Send-Token"),  
                      MessageTemplate.MatchPerformative(ACLMessage.INFORM));

    ACLMessage reply = myAgent.blockingReceive(mt);
    if (reply != null) {
        if (reply.getPerformative() == ACLMessage.INFORM) {
            tokenSig = reply.getContent();
            if (tokenSig.equals("Rejected")) {
                ACLMessage mg = new ACLMessage(ACLMessage.INFORM);
                mg.addReceiver (new AID("cryptoAgent", AID.ISLOCALNAME));
                mg.setConversationId("Finish-Encrypting");
                mg.setContent("Rejected|"+address+"|"+getLocalName());
                myAgent.send(mg);
                myAgent.doDelete();
                myAgent.doDelete();
                break;
            }
        }
    }
state++;
break;

case 2:
    String[] temp1 =null;
    temp1 = tokenSig.split("|");
    int index1 = tokenSig.indexOf("|");
    String token = tokenSig.substring(0,index1);
    String signature = tokenSig.substring(index1+1);
    //save token to a file
    byte[] TokenByte = Base64.decodeBase64(token.getBytes());
    File fToken=new File(agentName+"ReceivedToken.txt");
    try{
        FileOutputStream ft=new FileOutputStream(fToken);
        ft.write(TokenByte);
        ft.close();
    } catch(Exception e) {}  
    //save signature to a file
    byte[] SigByte = Base64.decodeBase64(signature.getBytes());`
File fSig = new File(agentName + "Sign.sig");
try{
    FileOutputStream fs = new FileOutputStream(fSig);
    fs.write(SigByte);
    fs.close();
} catch (Exception e) {

    // System.out.println();
    boolean sigToken = false;
    sigToken = dp.verifyToken(agentName);
    ACLMessage mg = new ACLMessage(ACLMessage.INFORM);
    mg.addReceiver(new AID("cryptoAgent", AID.ISLOCALNAME));

    if (sigToken == false) {
        System.out.println("The signature in the token is INVALID");
        // send message to cryptoAgent "Reject"
        mg.setConversationId("Finish-Encrypting");
        mg.setContent("Rejected|" + address + "|" + getLocalName());
        myAgent.send(mg);
        myAgent.doDelete();
    } else {
        System.out.println("The signature in the token is valid");

        boolean statusToken = false;
        statusToken = dp.decryptRand(TokenByte);

        if (statusToken == false) {
            System.out.println("Token is invalid");
            mg.setConversationId("Finish-Encrypting");
            mg.setContent("Rejected|" + address + "|" + getLocalName());
            myAgent.send(mg);
            myAgent.doDelete();
        } else {
            System.out.println("Token is valid");

            // send hashKey to MA...
            dp.createHashKey(agentName);

            // read hashkey.txt
            File fhashkey = new File(agentName + "hashkey.txt");
            byte[] bytehashkey = new byte[(int) fhashkey.length()];
            try {
                FileInputStream fhk = new FileInputStream(fhashkey);
                fhk.read(bytehashkey);
            } catch (Exception e) {
            }
            String strhashkey = new String(Base64.encodeBase64(bytehashkey));
            AID aid = new AID();
            aid.setName("test@" + resultAdd[1]);
            aid.addAddresses("http://" + address + ":7778/acc");
            ACLMessage tok = new ACLMessage(ACLMessage.INFORM);
            tok.setConversationId("The-Token");
        }
    }
}
tok.setContent(strhashkey);
tok.addReceiver (aid);
a.send(tok);
}
state++;
break;

case 3:
    mt1 = MessageTemplate.and(MessageTemplate.MatchConversationId("Finish-Encrypting"),MessageTemplate.MatchPerformative(ACLMessage.INFORM));
    ACLMessage reply1 = myAgent.blockingReceive(mt1);
    if (reply1 != null) {
        if (reply1.getPerformative() == ACLMessage.INFORM) {
            answer = reply1.getContent();
            state++;
        }
    }
    break;

case 4:
    ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
    msg.addReceiver (new AID("cryptoAgent", AID.ISLOCALNAME));
    msg.setConversationId("Finish-Encrypting");
    if (answer.equals("Finish"))
        msg.setContent("Finish|"+address+"|"+getLocalName());
    else
        msg.setContent("Rejected|"+address+"|"+getLocalName());
    myAgent.send(msg);

    finished = true;
    myAgent.doDelete();
    System.out.println(myAgent.getName() + " >> has terminated.. ");
    break;

} //switch...

private boolean finished = false;

public boolean done() {
    return finished;
}

} //End myBehaviour

}//end class SUA
public class MobileAgent extends Agent {
...
public void setup(){
    Object[] args = getArguments();
    ic = new infoController();
    ic = (infoController) args[0];
    agentName = (String) args[1]; // sua's name
    address = (String) args[2];
    suaName = (String) args[3]; // sua's name@address
    RAi = (String) args[4];
    ivbytes = (String) args[5];

    splitMe sm = new splitMe();
    hostAdd = sm.splitString(suaName, "@", 1);
    hostName = sm.splitString(hostAdd[1], ":", 1);
    pubKey = ic.getPubKey(); // get Kp

    String strMsg = null;
    byte[] fileContent = null;

    // contain ciphertext + Cipherkey + ciphercode
    File file = new File(agentName + "sendOver.txt");
    try{
        FileInputStream fin = new FileInputStream(file);
        fileContent = new byte[(int)file.length()];
        fin.read(fileContent);
        strMsg = new String(Base64.encodeBase64(fileContent));
    }catch(Exception e){}
    addBehaviour( new MyB(this, strMsg, pubKey));
} // setup()

private class MyB extends Behaviour{

    File fctext = new File(hostName[0] + "+agentName+ciphertext.txt");
    File fcode = new File(hostName[0] + "+agentName+decryptAgent.jar");

    Object[] argu = new Object[8];

    public MyB(Agent a, String msg, PrivateKey p){
        super(a);
        _msg = msg;
        _a = a;
        _state = 0;
        _done = false;
        priKey = p;
        argu[0] = pubKey; // Kp
    }
}
public void action(){

    switch(_state){

    case 0:
        System.out.println("\n\nMobileAgent "+getName()+ " is moving to "+address);
        AID a = new AID("amm@" + address + ":1099/JADE",true);
        a.addAddresses("http://" + address + ":7778/acc");
        PlatformID dest = new PlatformID(a);
        _state++;
        myAgent.doMove(dest);
        break;

    case 1:
        System.out.println("mobile agent migrated..");
        try{
            byteMsg = Base64.decodeBase64(_msg.getBytes());
            FileOutputStream fos = new FileOutputStream("sendO.txt");
            fos.write(byteMsg);
            fos.close();
        }catch(Exception ex){}
        System.out.println("saving a file....");
        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.addReceiver (new AID(RAi, AID.ISLOCALNAME));
        msg.setConversationId("Process-Message");
        myAgent.send(msg);

        mt = MessageTemplate.and(MessageTemplate.MatchConversationId("Process-Message"),MessageTemplate.MatchPerformative( ACLMessage.INFORM ));
        _state++;
        break;

    case 2:
        ACLMessage reply = myAgent.blockingReceive(mt);
        if (reply != null) {
            if (reply.getPerformative() == ACLMessage.INFORM) {
                String content = reply.getContent();
                if(!content.equals("Invalid")){
                    String[] temp1 =null;
                    temp1 = content.split("|");
                    int index1 = content.indexOf("|");
                    String ctext = content.substring(0,index1);
                    String codecd = content.substring(index1+1);
                    try{
                        byte[] ciptext = Base64.decodeBase64(ctext.getBytes());
                        byte[] ccode = Base64.decodeBase64(codecd.getBytes());
                        FileOutputStream fct =new FileOutputStream(fctext);
                        FileOutputStream fcd =new FileOutputStream(fcode);
                        fct.write(ciptext);
                        fct.close();
                        fcd.write(ccode);
                        fcd.close();
                    }catch(Exception ex){}
                }
            }
        }
    }
}
ACLMessage msg1 = new ACLMessage(ACLMessage.REQUEST);
msg1.addReceiver (new AID(RAi, AID.ISLOCALNAME));
msg1.setConversationId("Sign-Message");
myAgent.send(msg1);

try{ //un-jar Cd
    Runtime.getRuntime().exec("jar xf "+hostName[0]+"/"+agentName+"decryptAgent.jar
decryptAgent.class decryptAgent$validCodeEdit.class");
} catch(java.io.IOException e) {}
case 5:

    //get hashKey from SUA
    AID aid1 = new AID();
    aid1.setName(suaName);
    aid1.addAddresses("http://"+hostName[0]+":7778/acc");
    mt2 = MessageTemplate.and(MessageTemplate.MatchSender(aid1),
                                MessageTemplate.MatchPerformative(ACLMessage.INFORM));
    ACLMessage msg2 = myAgent.blockingReceive(mt2);

    if (msg2!=null){
        if (msg2.getPerformative() == ACLMessage.INFORM) {
            String hashkey = msg2.getContent();
            try {
                byte[] hkey = Base64.decodeBase64(hashkey.getBytes());

                FileOutputStream fhk = new FileOutputStream(fhashkey);
                fhk.write(hkey);
                fhk.close();
            } catch (Exception ex) {} //catch( Exception ex )

            String madName = getName(); //the MA’s name
            argu[1] = hkey;
            argu[2] = fctext;
            argu[3] = madName;
            argu[4] = address;
            argu[5] = hostName[0];
            argu[6] = RAi;
            argu[7] = ivbytes;

            createAgent(argu);
        }
    }
    _state++;
    break;

case 6:

    mt3 = MessageTemplate.and(MessageTemplate.MatchConversationId("Finish-Encrypting"), MessageTemplate.MatchPerformative(ACLMessage.INFORM));
    ACLMessage msg3 = myAgent.blockingReceive(mt3);

    //send result to SUA
    AID aid2 = new AID();
    aid2.setName(suaName);
    aid2.addAddresses("http://"+hostName[0]+":7778/acc");
    String content1 = msg3.getContent();

    ACLMessage finale = new ACLMessage(ACLMessage.INFORM);
    finale.addReceiver(aid2);
    finale.setConversationId("Finish-Encrypting");

    if(content1.equals("Finish"))
        finale.setContent("Finish");
else
    finale.setContent("Rejected");

    myAgent.send(finale);
    System.out.println(myAgent.getLocalName()+" has ended its itinerary");

    _done = true;
    myAgent.doDelete();
    break;

}//switch
}//action

public boolean done(){
    return _done;
}

//function to ask remote platform to create DA
public void createAgent(Object[] argu) {

    getContentManager().registerOntology(JADEManagementOntology.getInstance());
    getContentManager().registerOntology(FIPAManagementOntology.getInstance());
    Codec codec = new SLCodec();
    getContentManager().registerLanguage(codec, FIPANames.ContentLanguage.FIPA_SL);

    CreateAgent ca = new CreateAgent();
    ca.setAgentName("decrypt");
    String classname = "decryptAgent";
    ca.setClassName(classname);

    if (argu != null) {
        for (int i = 0; i < argu.length; i++) {
            ca.addArguments(argu[i]);
        }
    }

    ContainerID id=new ContainerID(AgentContainer.MAIN_CONTAINER_NAME, null);
    ca.setContainer(id);
    Action actExpr = new Action(getAMS(), ca);

    ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
    request.addReceiver(getAMS());
    request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
    request.setLanguage(codec.getName());
    request.setOntology(JADEManagementOntology.NAME);
    try {
        getContentManager().fillContent(request, actExpr);
        getContentManager().setValidationMode(true);
        send(request);
    } catch (Exception e) {e.printStackTrace();}

}//myB
}//class MobileAgentData
Table A9: DA class

**Function:** excerpts of DA’s behaviours

```java
public class decryptAgent extends Agent{
...

protected void setup(){
    Object[] args = getArguments();
    pubKey = (PublicKey) args[0];
    fhashkey = (File) args[1];
    fctext = (File) args[2];
    madName = (String) args[3];
    address = (String) args[4];
    hostName = (String) args[5];
    RAi = (String) args[6];
    ivbytes = (String) args[7];
    ivBytes = Base64.decodeBase64(ivbytes.getBytes());

    addBehaviour( new validCodeEdit( this ) );
}//setup

class validCodeEdit extends SimpleBehaviour {
...

    public validCodeEdit( Agent _a){
        super(_a);
        a = _a;
        state = 0;
        done1 = false;
        status = 1;

        // create RSA public key cipher
        try{
            pkCipher = Cipher.getInstance("RSA");
            pkCipher1 = Cipher.getInstance("RSA");
        }catch(Exception e){}
    }

    public void action() {
        switch(state){
            case 0:
                try{
                    //process hashKey and decrypt plaintext
                    hashp = processCiphertext(pubKey, fhashkey, fctext );
                }catch(java.lang.Exception e){}
                byte[] bytefp = new byte[(int)fplain.length()];
                String strMsg3 = null;
                String plainHash = null;

                try{
                    FileInputStream fp = new FileInputStream(fplain);
                    fp.read(bytefp);
                    strMsg3 = new String(Base64.encodeBase64(bytefp)); //hold plaintext
```
plainHash = strMsg3 + "|" + hashp;
}catch(Exception e){}

// Send request to check the plaintext to RA
ACLMessage msg3 = new ACLMessage(ACLMessage.REQUEST);
msg3.addReceiver(new AID(RAi, AID.ISLOCALNAME));
msg3.setConversationId("Check-PT");
msg3.setContent(plainHash);
myAgent.send(msg3);

mt3 = MessageTemplate.and(MessageTemplate.MatchConversationId("Check-PT"), MessageTemplate.MatchPerformative(ACLMessage.INFORM));
state++;
break;

case 1:
    ACLMessage reply3 = myAgent.blockingReceive(mt3);
    if (reply3 != null) {
        if (reply3.getPerformative() == ACLMessage.INFORM) {
            answer = reply3.getContent();//contain the status of check-pt
        }
    }
    state++;
    break;

case 2:
    AID aid = new AID();
    aid.setName(madName);
    aid.addAddresses("http://"+address+":7778/acc");

    ACLMessage msg2 = new ACLMessage(ACLMessage.INFORM);
    msg2.addReceiver(aid);
    msg2.setConversationId("Finish-Encrypting");

    if (answer.equals("true"))
        msg2.setContent("Finish");
    else
        msg2.setContent("Rejected");

    myAgent.send(msg2);

    System.out.println("" + myAgent.getName() + " has terminated.....");

    done1 = true;
    myAgent.doDelete();
    break;

}//switch
}//action

public boolean done(){
    return done1;
}
public String loadKey(File in, PublicKey Kp) throws GeneralSecurityException, IOException {

    pkCipher.init(Cipher.DECRYPT_MODE, Kp);
    byte[] cipherkeys = new byte [(int) in.length()];
    CipherInputStream is = new CipherInputStream(new FileInputStream(in), pkCipher);
    is.read(cipherkeys);

    String hkey = new String(Base64.decodeBase64(cipherkeys));

    //split key + info + H(P)
    String[] temp1 =null;
    temp1 = hkey.split("|");
    int index1 = hkey.indexOf("|");
    skey = hkey.substring(0,index1);//the key
    infohash = hkey.substring(index1+1);

    String[] tempo2 =null;
    tempo2 = infohash.split("|");
    int index2 = infohash.indexOf("|");
    info = infohash.substring(0,index2);//info = mlc
    hashp = infohash.substring(index2+1);//h(p)

    String[] tempor1 =null;
    tempor1 = info.split("-");
    int indexs1 = info.indexOf("-");  
    algo = info.substring(0,indexs1);

    if (algo.equals("AES")){
        if (keylengths == 256){
            try{
                aesKey = new byte[32];
                aesKey = Base64.decodeBase64(skey.getBytes());
                key = new SecretKeySpec(aesKey, "AES");
                aesCipher = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
            }catch (NoSuchAlgorithmException e) {}  
            catch (NoSuchProviderException e) {}  
            catch (NoSuchPaddingException e) {}  
        }else if (keylengths == 192) {
            try{
                aesKey = new byte[24];
                aesKey = Base64.decodeBase64(skey.getBytes());
                key = new SecretKeySpec(aesKey, "AES");
                aesCipher = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
            }catch (NoSuchAlgorithmException e) {}  
            catch (NoSuchProviderException e) {}  
            catch (NoSuchPaddingException e) {}  
        }else if (keylengths == 128) {
            try{
                aesKey = new byte[16];
            }

        }
    }else if (keylengths == 192) {
    try{
    }

    }else if (keylengths == 128) {
    try{
    }

    else if (keylengths == 256) {
    try{
    }

    }
}
aesKey = Base64.decodeBase64(skey.getBytes());
key = new SecretKeySpec(aesKey, "AES");
aesCipher = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
) catch (NoSuchAlgorithmException e){}
catch (NoSuchProviderException e){}
catch (NoSuchPaddingException e){}
}
else
    System.out.println("Only support AES 256/192/128...");
}/if
...
return hashp;
}

public void decrypt(File in, File out) throws IOException, InvalidKeyException {
    IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);
    try{
        aesCipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
    } catch(InvalidAlgorithmParameterException e){System.out.println(e);}

    CipherInputStream is = new CipherInputStream(new FileInputStream(in),
    aesCipher);
    FileOutputStream os = new FileOutputStream(out);
    copy(is, os);
    is.close();
    os.close();
}

private void copy(InputStream is, OutputStream os) throws IOException {
    int i;
    byte[] b = new byte[1024];
    while((i=is.read(b))!=-1) {
        os.write(b, 0, i);
    }
}

public String processCiphertext(PublicKey pubKey, File fhashkey, File fctext
 ) throws Exception{

    File fplain = new File(hostName+"/plaintext.txt");
    String hashp = null;
    try{
        hashp = loadKey(fhashkey, pubKey);
        decrypt(fctext,fplain);
    } catch(GeneralSecurityException ioe){ }
    catch(Exception ioe){ }
    return hashp;
}
}//validCode
}//class decryptAgent
public class receiverAgent extends Agent {
    ...
    protected void setup() {
        MLC m = new MLC(comlayer);
        String mlcK2 = m.getMLC();

        try{
            rm = new receiveMessage(mlcK2);
        } catch(GeneralSecurityException ioe){ } catch(Exception ioe){ }
        // Add the behaviour serving request
        addBehaviour(new Processing());
        addBehaviour(new Signing());
        addBehaviour(new Verifyhp());
    }

    private class Processing extends CyclicBehaviour {
        ...
        protected void action() {
            MessageTemplate mt =
                MessageTemplate.and(MessageTemplate.MatchConversationId("Process-Message"),
                MessageTemplate.MatchPerformative(ACLMessage.REQUEST));
            ACLMessage msg = myAgent.blockingReceive(mt);
            if (msg != null) {
                String messages = msg.getContent();
                sender = msg.getSender().getName();
                splitMe sm = new splitMe();
                String result[] = sm.splitString(sender,":",1);
                hostName = sm.splitString(result[1],":",1);//take hostName[0] as hostname.
                new File(hostName[0]).mkdir();
                File fromMA = new File("sendO.txt");
                File sendO = new File(hostName[0]+"/sendOver.txt");
                FileRead fr = new FileRead();//copy file to the created directory
                try{
                    fr.copyFile(fromMA,sendO);
                } catch (Exception e){}

                //process the received message
                //status indicates that both S and H(Cd) are valid/Invalid
                boolean status =false;
                try{
                    status = rm.receiveMsg(hostName[0]);
                } catch(GeneralSecurityException ioe){ }
        }
    }
}
catch(Exception ioe){ }

if(status == true){
    File fctext = new File(hostName[0]+"/ciphertext.txt");
    File fcode = new File(hostName[0]+"/decryptAgent.jar");
    String signToken = null;
    byte[] readctext = new byte[(int) fctext.length()];
    byte[] readcode = new byte[(int) fcode.length()];
    try {
        FileInputStream fct = new FileInputStream(fctext);
        fct.read(readctext);
        FileInputStream fcd = new FileInputStream(fcode);
        fcd.read(readcode);
    } catch (Exception e) {} 
    String strctext = new String(Base64.encodeBase64(readctext));
    String strcode = new String(Base64.encodeBase64(readcode));
    String ctextCd = strctext + "|" + strcode;
    ACLMessage reply = msg.createReply();
    reply.setPerformative(ACLMessage.INFORM);
    reply.setContent(ctextCd);
    myAgent.send(reply);
} //if
else{
    ACLMessage reply = msg.createReply();
    reply.setPerformative(ACLMessage.INFORM);
    reply.setContent("Invalid");
    myAgent.send(reply);
    myAgent.doDelete();
}
} // End of inner class

private class Signing extends CyclicBehaviour {

    public void action() {

        AID aid = new AID();
        aid.setName(sender);
        aid.addAddresses("http://"+hostName[0]+":7778/acc");
        MessageTemplate mt1 = MessageTemplate.and(MessageTemplate.MatchSender(aid),
                                                  MessageTemplate.MatchConversationId("Sign-Message"));
        ACLMessage msg1 = myAgent.blockingReceive(mt1);

        if (msg1 != null) {
            File fToken = new File(hostName[0]+"/Token.txt");
            String signToken = null;
            try{
                signToken = rm.Signing(fToken);
            } catch(GeneralSecurityException ioe){ }
        }
    }
} // End of inner class

278
catch(Exception ioe){ }

byte[] readToken = new byte[(int) fToken.length()];
try {
    FileInputStream ft = new FileInputStream(fToken);
    ft.read(readToken);
} catch (Exception e) {System.out.println("File Not Found.");

String strToken = new String(Base64.encodeBase64(readToken));
String tokenSig = strToken + "|" + signToken;
ACLMessage reply1 = msg1.createReply();
reply1.setPerformative(ACLMessage.INFORM);
reply1.setContent(tokenSig);
myAgent.send(reply1);
}
}  // End of inner class
private class Verifyhp extends CyclicBehaviour {
...

public void action() {
    MessageTemplate mt3 =
        MessageTemplate.and(MessageTemplate.MatchConversationId("Check-PT"),
                              MessageTemplate.MatchPerformative(ACLMessage.REQUEST));

    ACLMessage msg3 = myAgent.blockingReceive(mt3);

    if (msg3 != null) {
        String messages3 = msg3.getContent();
        //split plain and hash
        String[] temp1 =null;
        temp1 = messages3.split("|");
        int index1 = messages3.indexOf("|");
        String plain = messages3.substring(0,index1);
        String hashplain = messages3.substring(index1+1);

        byte[] byteMsg3 =null;
        File ff = new File(hostName[0]+"/plaintext1.txt");
        try{
            byteMsg3 = Base64.decodeBase64(plain.getBytes());
            FileOutputStream fos3 = new FileOutputStream(ff);
            fos3.write(byteMsg3);
            fos3.close();
        } catch(Exception ex){System.out.println("FileNotFoundException : " + ex);}

        boolean statusMD= false;
        try{
            statusMD = rm.recomputeMD(ff,hashplain);
        } catch(Exception ioe){ }

        if (statusMD == true){
            System.out.println("Genuine Message received.");
            ACLMessage reply3 = msg3.createReply();
            reply3.setPerformative(ACLMessage.INFORM);
            reply3.setContent("true");
        }
myAgent.send(reply3);
}
else{
    System.out.println("Message has been tampered!.");
    ACLMessage reply3 = msg3.createReply();
    reply3.setPerformative(ACLMessage.INFORM);
    reply3.setContent("false");
    myAgent.send(reply3);
    myAgent.doDelete();
}
} // End of inner class

Table A11: receiveMessage class

**Function:** Excerpts programs to execute processes at the recipient’s side.

```java
public class receiveMessage{
    ....
    byte[] ivBytes = null;
    String mlc = null;

    public receiveMessage(String mlc1) throws GeneralSecurityException {
        status = 0;
        pkCipher = Cipher.getInstance("RSA");
        pkCipher1 = Cipher.getInstance("RSA");
        mlc = mlc1;
    }

    //decrypt the cipherkey that contains H(Cd) and K2...
    public String loadKey(File in, PrivateKey recPriKey, int symkey) throws GeneralSecurityException, IOException {
        String hashcode =null;
        pkCipher.init(Cipher.DECRYPT_MODE, recPriKey);
        byte[] cipherkeys = new byte [(int) in.length()];
        CipherInputStream is = new CipherInputStream(new FileInputStream(in),
        pkCipher);
        is.read(cipherkeys);
        String hashandkey = new String(Base64.decodeBase64(cipherkeys));
        String[] temp1 =null;
        temp1 = hashandkey.split("|") ;
        int index1 = hashandkey.indexOf("|");
        String skey = hashandkey.substring(0,index1);
        String hashcode1 = hashandkey.substring(index1+1);
        String[] temp2 =null;
        temp2 = hashcode1.split("|");
        int index2 = hashcode1.indexOf("|");
        String SivBytes = hashcode1.substring(0,index2);
        hashcode = hashcode1.substring(index2+1);
```
ivBytes = Base64.decodeBase64(SivBytes.getBytes());

//split mlc
String[] tempor1 =null;
tempor1 = mlc.split("-");
int indexs1 = mlc.indexOf("-");
algo = mlc.substring(0,indexs1);
String kmp = mlc.substring(indexs1+1);
...
String keylength = kmp.substring(0,indexs2);
String mp = kmp.substring(indexs2+1);
...
int keylengths = Integer.parseInt(keylength);
if (algo.equals("AES")){status = 1;
  if (keylengths == 256){
    System.out.println("MLC: AES/256 ");
    System.out.println();
    try{
      aesKey = new byte[32];
      aesKey = Base64.decodeBase64(skey.getBytes());
      key = new SecretKeySpec(aesKey, "AES");
      aesCipher = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
    }catch (Exception e){}
  }
else if (keylengths == 192) {status = 2;
  System.out.println("MLC: AES/192 ");
  try{
    aesKey = new byte[24];
    aesKey = Base64.decodeBase64(skey.getBytes());
    key = new SecretKeySpec(aesKey, "AES");
    aesCipher = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
  }catch (Exception e){}
} else if (keylengths == 128) {status = 3;
  System.out.println("MLC: AES/128 ");
  try{
    aesKey = new byte[16];
    aesKey = Base64.decodeBase64(skey.getBytes());
    key = new SecretKeySpec(aesKey, "AES");
    aesCipher = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
  }catch (Exception e){}
}...
return hashcode;
}

//Decrypts 'in' while copies the data in 'out' using K2
public void decrypt(File in, File out) throws IOException, InvalidKeyException {
  IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);
  try{
    aesCipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
  }catch(InvalidAlgorithmParameterException e){System.out.println(e);}
  CipherInputStream is = new CipherInputStream(new FileInputStream(in),

aesCipher;
FileOutputStream os = new FileOutputStream(out);
copy(is, os);
is.close();
os.close();
}

private void copy(InputStream is, OutputStream os) throws IOException {
...
}

public String Signing(File code) throws GeneralSecurityException, IOException {
byte[] signCd = new byte[(int) code.length()];
try {
    FileInputStream fileInputStream = new FileInputStream(code);
    fileInputStream.read(signCd);
} catch (Exception e) {}{
Signature sig = Signature.getInstance("SHA1withRSA");
sig.initSign(recPriKey);
sig.update(signCd);
byte[] byteSignCd = sig.sign();
String byteSignStr = new String(Base64.encodeBase64(byteSignCd));
return byteSignStr;
}

boolean recomputeMD(File fstrCd, String hashcode){
byte[] ByteCd = new byte[(int) fstrCd.length()];
try {
    FileInputStream fileMD = new FileInputStream(fstrCd);
    fileMD.read(ByteCd);
} catch (Exception e) {}{
byte[] newMsgDigest = null;
try{
    MessageDigest md = MessageDigest.getInstance("SHA1");
    md.update(ByteCd);
    newMsgDigest = md.digest();
} catch (NoSuchAlgorithmException e){}
String strNewMsgDigest = new String(Base64.encodeBase64(newMsgDigest));
boolean statusMD = strNewMsgDigest.equals(hashcode);
if(!strNewMsgDigest.equals(hashcode))
    System.out.println("P has been tampered!");
else
    System.out.println("Genuine P received.");
return statusMD;
}

public boolean receiveMsg(String hostname) throws Exception{
// load keystore and truststore
...
File freceived = new File(hostname + "/sendOver.txt");
byte[] byteAll = new byte[(int) freceived.length()];
try {
FileInputStream fileAll = new FileInputStream(freceived);
fileAll.read(byteAll);
} catch (FileNotFoundException e) {}  
String decodedMsgAll = new String(Base64.decodeBase64(byteAll));

// ciphertext
String[] temp1 = null;
temp1 = decodedMsgAll.split("|");

int index1 = decodedMsgAll.indexOf("|");
String ciphertext = decodedMsgAll.substring(0, index1);
String decodedMsgAll1 = decodedMsgAll.substring(index1 + 1);

// Ciphercode
String[] temp2 = null;
temp2 = decodedMsgAll1.split("|");
int index2 = decodedMsgAll1.indexOf("|");
String ciphercode = decodedMsgAll1.substring(0, index2);
String decodedMsgAll2 = decodedMsgAll1.substring(index2 + 1);

// == save ciphertext, Cipherkey, and Ciphercode to files
...

// use K2 to decrypt ciphercode
File stringCiphercode = new File(hostname + "/stringciphercode.txt");
decrypt(fciphercode, stringCiphercode);

// read stringciphercode.txt and split to get Cd, S, T.
byte[] readCiphercode = new byte[(int) stringCiphercode.length()];
try {
    FileInputStream fc = new FileInputStream(stringCiphercode);
fcc.read(readCiphercode);
} catch (Exception e) {}  
String stringciphercode64 = new String(Base64.decodeBase64(readCiphercode));

// strCd
String[] temp1 = null;
temp1 = stringciphercode64.split("|");
int indexs1 = stringciphercode64.indexOf("|");
String strCd = stringciphercode64.substring(0, indexs1);
String stringciphercode641 = stringciphercode64.substring(indexs1 + 1);

// theSignature
String[] temp2 = null;
temp2 = stringciphercode641.split("|");
int indexs2 = stringciphercode641.indexOf("|");
String theSignature = stringciphercode641.substring(0, indexs2);
String stringciphercode642 = stringciphercode641.substring(indexs2 + 1);

// save strCd, S and T to files
byte[] strCdByte = Base64.decodeBase64(strCd.getBytes());
byte[] theSignatureByte = Base64.decodeBase64(theSignature.getBytes());
byte[] TokenByte = Base64.decodeBase64(stringciphercode642.getBytes());
...
boolean StatusAll = false;
//verify signature
boolean signatureVerified=false;
VerifySignatureX vSig= new VerifySignatureX();
signatureVerified = vSig.vSignature(senPubKey,hostname);
if ( signatureVerified == false)
    System.out.println("Sender's Signature is not valid!");
else
    System.out.println("The signature is valid, message is from sender");

boolean statusMd = recomputeMD(fstrCd,hashcode);
if(statusMd == true)
    System.out.println("Genuine Message received.");
else
    System.out.println("Message has been tampered!");

if ((signatureVerified==true) && (statusMd==true))
    StatusAll = true;
return StatusAll;
}
public class sSua extends Agent{

    private StringBuffer strtosend = null;

    protected void setup() {
        engine = new CryptoEngine ();
        try{
            aestry = new AESFast(this);
        }catch (Exception e) {e.printStackTrace();}
    }

    public void getMessage(final String msg){
        addBehaviour(new OneShotBehaviour() {
            public void action() {
                try{
                    dp = new dataPrep1(aestry);
                }catch (Exception e) {e.printStackTrace();}
                msgs = msg;
                str = split(msgs, '|');
                StringBuffer strtosend = preparedata (str[1]);
                addBehaviour(new SendMessage(strtosend));
                ...
            }//action
        });
    }

    public StringBuffer preparedata (String str) {//str = text to be encrypted
        try{
            dp.mainStart();
            ciphertext1 = dp.getCiphertext();
            int ciphertextlen = ciphertext1.length();
            cipherkey= dp.getCipherkey();
            int cipherkeylen = cipherkey.length();
            ciphercode = dp.getCiphercode();
            int ciphercodelen = ciphercode.length();
            sendOv = new StringBuffer(ciphertextlen+cipherkeylen+ciphercodelen+3);
        }catch (Exception e) {e.printStackTrace();}
        sendOv.append(cipherkey);
        sendOv.append("|");
        sendOv.append(ciphertext1);
        sendOv.append("|");
        sendOv.append(ciphercode);
        return sendOv;
    }//preparedata

    public String[] split( String str, char separatorChar ) {
        if ( str == null ) {
            return null;
        }
```java
return null;

int len = str.length();
if (len == 0) {
    return null;
}
Vector list = new Vector();
int i = 0;
int start = 0;
boolean match = false;
while (i < len) {
    if (str.charAt(i) == separatorChar) {
        if (match) {
            list.addElement(str.substring(start, i).trim());
            match = false;
        }
        start = ++i;
        continue;
    }
    match = true;
    i++;
} 
if (match) {
    list.addElement(str.substring(start, i).trim());
}
String[] arr = new String[list.size()];
list.copyInto(arr);
return arr;
}

class SendMessage extends Behaviour {
...

SendMessage(StringBuffer mesej) {
    str = mesej.toString();
    _done = false;
    _state = 0;
}
public void action() {
    switch (_state) {
    case 0:
        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.addReceiver(new AID("CLA", AID.ISLOCALNAME));
        msg.setConversationId("Send-Message");
        msg.setContent("Mobile");
        myAgent.send(msg);
        break;
    case 1:
        mt = MessageTemplate.and(MessageTemplate.MatchConversationId("Send-
Message"), MessageTemplate.MatchPerformative(ACLMessage.REQUEST)
ACLMessage ms = myAgent.blockingReceive(mt);
        if (ms != null) {
            String m = ms.getContent();//Agree|sRA-name |mlc
```
str1 = split(m, '('|');

if(str1[0].equals("AGREE")){
    ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
    msg.addReceiver (new AID("sRA", AID.ISLOCALNAME));
    msg.setConversationId("Proc-Message");
    msg.setContent(str);
    myAgent.send(msg);
    _state++;
    break;
}

case 2:
    mt = MessageTemplate.and(MessageTemplate.MatchConversationId("Proc-
Message"), MessageTemplate.MatchPerformative( ACLMessage.INFORM ));
    ACLMessage msg1 = myAgent.blockingReceive(mt);
    if (msg1 != null) {
        String m = msg1.getContent();
        if ( m.equals("VALID") ) {
            ACLMessage msg2 = new ACLMessage(ACLMessage.REQUEST);
            msg2.addReceiver (new AID("sRA, AID.ISLOCALNAME"));
            msg2.setConversationId("Create-agent");
            myAgent.send(msg2);
        }
    }
    _state++;
    break;

case 3:
    mt1 = MessageTemplate.and(MessageTemplate.MatchConversationId("Send-
Token"), MessageTemplate.MatchPerformative( ACLMessage.REQUEST ));
    ACLMessage msg3 = myAgent.blockingReceive(mt1);
    if (msg3 != null) {
        if (msg3.getPerformative() == ACLMessage.REQUEST) {
            sendToken = msg3.getContent();
            boolean checkTok = dp.checkToken(sendToken);
            ...
            //create hashkey ...
            String hashKey = dp.getHashKey();
            ACLMessage reply = msg3.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent(hashKey);
            myAgent.send(reply);
            System.out.println("> Sending hashKey to DA ");
        }
    }
    else
        System.out.println("Null message from DA");
    _state++;
    break;

case 4:
    mt2 = MessageTemplate.and(MessageTemplate.MatchConversationId("Finish-
Decrypting"),MessageTemplate.MatchPerformative( ACLMessage.INFORM ));
    ACLMessage msg4 = myAgent.blockingReceive(mt2);
if (msg4 != null){
    String answ = msg4.getContent();
    if (answ.equals("Finish"))
        System.out.println("sDA has finished encrypting");
    else
        myAgent.doWait(RETRY_TIME);
    myAgent.addBehaviour(reSend(strtosend));
}
//if
_done = true;
break;
}//switch
}//action

public boolean done(){
    return _done;
}
}//inner class
}//class ssua

Table A13: sDA class

**Function:** excerpts of sDA’s behaviours

public class decryptAgent extends Agent {

    protected void setup() {

        Object[] args = getArguments();
        deviceAdd = (String) args[0];
        ciphertext = (String) args[1];
        setName = (String) args[2];
        bs = new Base64();

        byte fileContent1[] = null;
        byte fileContent2[] = null;
        File file1 = new File("RSAmod.dat");
        File file2 = new File("RSApubExp.dat");

        try {
            FileInputStream fin1 = new FileInputStream(file1);
            FileInputStream fin2 = new FileInputStream(file2);
            fileContent1 = new byte[(int)file1.length()];
            fileContent2 = new byte[(int)file2.length()];

            fin1.read(fileContent1);
            fin2.read(fileContent2);

            BigInteger RSAmod = new BigInteger(fileContent1);
            BigInteger RSApubExp = new BigInteger(fileContent2);
            fin1.close();
            fin2.close();
        }
    }
}
pubKey = new RSAKeyParameters(false, RSAmod, RSApubExp);
}catch(Exception e){System.out.println("File not found" + e);}

addBehaviour(new decryptMsg(this));
}

class decryptMsg extends Behaviour {
...

decryptMsg(Agent _a) {
    a = _a;
    _done =false;
    _state = 0;
}

public void action() {
    switch(_state){
    case 0:
        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.addReceiver (new AID("sRA", AID.ISLOCALNAME));
        msg.setConversationId("Sign-Message");
        myAgent.send(msg);
        _state++;
        break;
    case 1:
        MessageTemplate mt1 = MessageTemplate.and
        (MessageTemplate.MatchConversationId ("Sign-Message"),
        MessageTemplate.MatchPerformative(ACLMessage.INFORM));
        ACLMessage msg1 = myAgent.blockingReceive(mt1);
        if (msg1 != null) {
            String sigToken = msg1.getContent();
            AID aid = new AID();
            aid.setName(setName);
            aid.addAddresses("jicp://+deviceAdd"+deviceAdd); //send to sSUA
            ACLMessage tok = new ACLMessage(ACLMessage.REQUEST);
            tok.addReceiver (aid);
            tok.setConversationId("Send-Token");
            tok.setContent(sigToken);
            myAgent.send(tok);
        }
        _state++;
        break;
    case 2:
        MessageTemplate mt2 = MessageTemplate.and
        (MessageTemplate.MatchConversationId("Send-Token"),
        MessageTemplate.MatchPerformative(ACLMessage.INFORM));
        ACLMessage msg2 = myAgent.blockingReceive(mt2);
        if (msg2 != null) {
            String hashKey1 = msg2.getContent();
            byte[] todecrypt = bs.decode(hashKey1);
byte[] hashkeybyte = null;
try{
    hashkeybyte = RSADecrypt(todecrypt);
}catch(Exception e){System.out.println(e);};

String hashKey = new String (hashkeybyte);
String delimiter = "|";

String[] temp1 =null;
temp1 = hashKey.split(delimiter);
int index1 = hashKey.indexOf(delimiter);
String digestP = hashKey.substring(0,index1);

bytecipher = (bs.decode(ciphertext));

String temp = hashKey.substring(index1+1);
String[] temp2 =null;
temp2 = temp.split(delimiter);
     int index2 = temp.indexOf(delimiter);
String mlc = temp.substring(0,index2);

String keyIV = temp.substring(index2+1);
bytekey1 = new KeyParameter (bs.decode(keyIV));
plaintext = calldecrypt(mlc, bytecipher);
}
//end if
_state++;break;

case 3:
    ACLMessage msg3 = new ACLMessage(ACLMessage.REQUEST);
    msg3.addReceiver (new AID("receiver", AID.ISLOCALNAME));
    msg3.setConversationId("Check-PT");
    myAgent.send(msg3);
    _state++;break;

case 4:
    MessageTemplate mt4 = MessageTemplate.and
    (MessageTemplate.MatchConversationId("Check-PT"),
    MessageTemplate.MatchPerformative(ACLMessage.INFORM));

    ACLMessage msg4 = myAgent.blockingReceive(mt4);
    String cPT = msg4.getContent();

    AID aid1 = new AID();
    aid1.setName(setName);
    aid1.addAddresses("jicp://"+deviceAdd);
    ACLMessage tok1 = new ACLMessage(ACLMessage.INFORM);
tok1.addReceiver (aid1);
tok1.setConversationId("Finish-Decrypting");

    if (cPT != null) {
        if(msg4.equals("Valid"))
tok1.setContent("Finish");

else
    tok1.setContent("Reject");
    myAgent.send(tok1);
//end if
    _done = true;
    break;

 }//switch
}//action

public boolean done(){
    return _done;
}

class BLODecrypt {

    public byte[] BLODecrypt(byte[] data) throws Exception {
        cipher.init(false, bytekey1);
        int size = cipher.getOutputSize(data.length);
        byte[] result = new byte[size];
        int len = cipher.processBytes(data, 0, data.length, result, 0);

        if(data==null || data.length==0)
            return new byte[0];

        len += cipher.doFinal(result, len);
    }
}
if(len < size){
    byte[] tmp = new byte[len];
    System.arraycopy(result,0,tmp,0,len);
    result = tmp;
}
return result;
}
}
//decryptMessage
}
//decryptAgent

Table A14: sRA class

**Function**: excerpts of sRA's behaviours

```java
public class sRA extends Agent {

    ... protected void setup() {
        cipher = new PaddedBlockCipher(new CBCBlockCipher( new BlowfishEngine() ) );
        // create pub key for Bob, either
        byte fileContent1[] = null;
        byte fileContent2[] = null;

        File file1 = new File("RSAmod1.dat");
        File file2 = new File("RSApubExp1.dat");
        try{
            FileInputStream fin1 = new FileInputStream(file1);
            FileInputStream fin2 = new FileInputStream(file2);
            fileContent1 = new byte[(int)file1.length()];
            fileContent2 = new byte[(int)file2.length()];
            fin1.read(fileContent1);
            fin2.read(fileContent2);

            BigInteger RSAmod = new BigInteger(fileContent1);
            BigInteger RSApubExp = new BigInteger(fileContent2);
            fin1.close();
            fin2.close();

            BobRSApubKey = new RSAKeyParameters(false, RSAmod, RSApubExp);
        }catch(FileNotFoundException e){System.out.println("File not found" + e);}
        catch(IOException ioe){System.out.println("Exception while reading the file " + ioe);}

        addBehaviour(new service1(this));
    }
    //setup

    class service1 extends Behaviour {
```
public service1(Agent _a) {
    a = _a;
    _done = false;
    _state = 0;
}

public void action() {
    switch (_state) {
        case 0:
            MessageTemplate mt1 =
                MessageTemplate.and(MessageTemplate.MatchConversationId("Proc-
                    Message"), MessageTemplate.MatchPerformative(ACLMessage.INFORM));
            ACLMessage msg1 = myAgent.blockingReceive(mt1);

            if (msg1 != null) {
                messages = msg1.getContent();
                System.out.println("gettin' started...");
                setName = msg1.getSender().getName();
                String deviceTemp = msg1.getSender().toString();
                System.out.println("add of device: "+setName+" getSender(): "+deviceTemp);

                splitMe splitme = new splitMe();
                String add1[] = splitme.splitString(setName,"@",1);
                String address1 = add1[1];

                String add2[] = splitme.splitString(address1,"/",1);
                deviceAdd = add2[0];
            }

        splitMe sm = new splitMe();
        String splitMessage[] = sm.splitString(messages,"|",2);

        String s1 = splitMessage[0];
        byte[] bytecipherkey = (bs.decode(s1));

        ciphertext = splitMessage[1];//cipherkey (decrypt with privKr)

        String s2 = splitMessage[2];//ciphercode decrypt with K2
        byte[] bytecodeciphercode = (bs.decode(s2));

        try{
            KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
            char[] password = pwd.toCharArray();
            File fileKS = new File (myFile);

            FileInputStream fks = new FileInputStream(fileKS);
            ks.load(fks, password);
            fks.close();

            char[] pwd = pwd1.toCharArray();
            Key myKey = ks.getKey(myName,pwd1);//name of the owner's keystore
            senderPriKey = (RSAPrivateCrtKey) myKey;
        } catch (Exception e) {
            System.out.println("Exception: "+e.getMessage());
        }
    }
}
BigInteger RSAgetModulus = senderPriKey.getModulus();
BigInteger RSAgetPublicExponent = senderPriKey.getPublicExponent();
BigInteger RSAgetPrivateExponent = senderPriKey.getPrivateExponent();
BigInteger RSAgetPrimeP = senderPriKey.getPrimeP();
BigInteger RSAgetPrimeQ = senderPriKey.getPrimeQ();
BigInteger RSAgetPrimeExponentP = senderPriKey.getPrimeExponentP();
BigInteger RSAgetPrimeExponentQ = senderPriKey.getPrimeExponentQ();
BigInteger RSAgetCrtCoefficient = senderPriKey.getCrtCoefficient();

cPrivateKey = new RSAPrivateCrtKeyParameters
  (RSAgetModulus, RSAgetPublicExponent, RSAgetPrivateExponent, RSAgetPrimeP, RSAgetPrimeQ, RSAgetPrimeExponentP, RSAgetPrimeExponentQ, RSAgetCrtCoefficient);

} catch (Exception e) {System.out.println(e);}

byte[] result = null;

try{
  result = RSADecrypt(bytecipherkey);
} catch (Exception e3){}

String strCipherkey = new String(result); // => digestCd + "|" + keyIV;
splitMe sml = new splitMe(); // split to get digestCd and keyIV
String splitMessage1[] = sml.splitString(strCipherkey, "|", 1);
String digestCd = splitMessage1[0];
String KEYiv = splitMessage1[1];

// FOR BLO:
bytekey1 = new KeyParameter (bs.decode(KEYiv));

// decrypt ciphercode (s1) with K2..
byte[] decryptCiphercode = null;
try{
  decryptCiphercode = BLODecrypt(byteciphercode);
} catch (Exception e) { System.out.println(e); }

String ciphercode = new String(decryptCiphercode);

splitMe sm3 = new splitMe(); // split to get the AESkey and IV for (K2)
String splitMessage3[] = sm3.splitString(ciphercode, "|", 2);
token = splitMessage3[0]; // token..

String strSiggy = splitMessage3[1]; // signature of Cd
byte[] byteSiggy = (bs.decode(strSiggy));

String strCd = splitMessage3[2]; // the Cd
byte[] bytecd = (bs.decode(strCd));

try{
  FileOutputStream fos = new FileOutputStream("decryptAgent1.dat")
  fos.write(bytecd);
  fos.close();
System.gc();
File oldfile = new File("decryptAgent1.dat");
File newfile = new File("decryptAgent1.jar");
boolean rename = oldfile.renameTo (newfile);

}catch(Exception ex){}
try{
    rf = new RenameFile();
}catch(Exception ioe){ System.out.println("IOException : " + ioe);}
byte fileContent[] = null;
String s = null;

boolean resultTF=false;
try{
    resultTF = RSAVerify(bytecd, byteSiggy);
    if(resultTF==true)
        System.out.println("Signature valid");
    else
        System.out.println("Signature INVALID");
}catch(Exception e){}

//verify hash... compare with String digestCd
byte[] mdigest = computeMD(bytecd);
String newMdigest = new String(bs.encode(mdigest));
System.out.println("New Hash(CD): "+newMdigest);

boolean tf = false;
if(!newMdigest.equals(digestCd))
    System.out.println("Cd is tampered");
else{
    System.out.println("Cd is genuine");
    ACLMessage reply = msg1.createReply();
    reply.setPerformative(ACLMessage.INFORM);
    reply.setContent("VALID");

    myAgent.send(reply);
    System.gc();
    rf.renaming(f1,newFile1);
    rf.renaming(f2,newFile2);
    rf.rejar();
}
_state++;
break;

case 1:
    rf.unjar();
    MessageTemplate mt =
    MessageTemplate.and(MessageTemplate.MatchConversationId("Create-agent"),MessageTemplate.MatchPerformative(ACLMessage.REQUEST ));
    ACLMessage msg2 = myAgent.blockingReceive(mt);
if (msg2 != null) {
    System.gc();
    Object[] args1 = new Object[3];
    args1[0] = deviceAdd;
    args1[1] = ciphertext;
    args1[2] = setName;
    System.out.println("\n deviceAdd: "+deviceAdd);
    AgentContainer c = getContainerController();
    try {
        AgentController ad = c.createNewAgent( "sDA", "decryptAgent",
            args1 );
        ad.start();
    }catch (Exception e){System.out.println(e);} 
}
_state++;
break;
case 2:
    MessageTemplate mtemp1 =
    MessageTemplate.and(MessageTemplate.MatchConversationId("Sign-
        Message"),MessageTemplate.MatchPerformative( ACLMessage.REQUEST ));
    msg4 = myAgent.blockingReceive(mtemp1);
    if (msg4 != null) {
        String sToken = null;
        try{
            byte[] tokenbyte = token.getBytes();
            signature = RSASign(tokenbyte);

            sig = new String (bs.encode(signature));
            String strToken = sig + "|"+token;

            byte[] sendToken = strToken.getBytes();
            sToken = new String(bs.encode(sendToken));
        }catch(Exception e){System.out.println(e);} 

        ACLMessage reply4 = msg4.createReply();
        reply4.setPerformative(ACLMessage.INFORM);
        reply4.setContent(sToken);
        myAgent.send(reply4);
    } //if
    _state++;
    break;
case 3:
    MessageTemplate mtemp5 = MessageTemplate.and
    (MessageTemplate.MatchConversationId ("Check-PT"),
    MessageTemplate.MatchPerformative( ACLMessage.REQUEST ));
    ACLMessage msg5 = myAgent.blockingReceive(mtemp5);
    if (msg5 != null) {
        ACLMessage reply5 = msg5.createReply();
        reply5.setPerformative(ACLMessage.INFORM);
myAgent.send(reply5);
}

_done = true;
break;
} // switch
} // action

public boolean done()
{
    return _done;
}

public byte[] computeMD(byte[] S){
    Digest dig = new SHA1Digest();
    byte[] byteMD = S;
    dig.update(byteMD, 0, byteMD.length);
    byte[] digValue = new byte[dig.getDigestSize()];
    dig.doFinal(digValue, 0);
    return digValue;
}

public byte[] RSASign (byte[] toSign) throws Exception {
    if (caPrivateKey == null)
        throw new Exception("Generate RSA keys first!");
    SHA1Digest dig = new SHA1Digest();
    RSAEngine eng = new RSAEngine();
    PSSSigner signer = new PSSSigner(eng, dig, 64);
    signer.init(true, caPrivateKey);
    signer.update(toSign, 0, toSign.length);
    return signer.generateSignature();
}

public byte[] RSADecrypt (byte[] toDecrypt) throws Exception {
    if (caPrivateKey == null)
        throw new Exception("Generate RSA keys first!");
    AsymmetricBlockCipher eng = new RSAEngine();
    eng = new PKCS1Encoding(eng);
    eng.init(false, caPrivateKey);
    return eng.processBlock(toDecrypt, 0, toDecrypt.length);
}

public boolean RSAVerify (byte[] msg, byte[] sig) throws Exception {
    if (BobRSApubKey == null)
        throw new Exception("Generate RSA keys first before verifying!");
    SHA1Digest dig = new SHA1Digest();
    RSAEngine eng = new RSAEngine();
    PSSSigner signer = new PSSSigner(eng, dig, 64);
    signer.init(false, BobRSApubKey);
signer.update(mesg, 0, mesg.length);
return signer.verifySignature(sig);
}

public byte[] BLODecrypt( byte[] data) throws Exception {
    cipher.init( false, bytekey1 );
    int size = cipher.getOutputSize( data.length );
    byte[] result = new byte[ size ];
    int len = cipher.processBytes(data,0,data.length,result,0 );

    if(data == null || data.length == 0){
        return new byte[0];
    }
    len += cipher.doFinal(result,len);

    if(len < size){
        byte[] tmp = new byte[olen];
        System.arraycopy(result,0,tmp,0,len);
        result = tmp;
    }

    return result;
}

}//end class behaviour
}//end class receiver
public class Code {
    ...
    public Code(PublicKey pubk){//Kp is read from file
        pubKey = pubk;
    }

    public String loadKey(File in, PublicKey Kp) throws GeneralSecurityException, IOException {
        String hashandkey = null;
        String infohash = null;
        String info = null;
        String hashp = null;

        pkCipher.init(Cipher.DECRYPT_MODE, Kp);
        byte[] cipherkeys = new byte[(int) in.length()];
        CipherInputStream is = new CipherInputStream(new FileInputStream(in), pkCipher);
        is.read(cipherkeys);
        String hkey = new String(Base64.decodeBase64(cipherkeys));

        //split key + info + H(P)
        String[] temp1 =null;
        temp1 = hkey.split("|");
        int index1 = hkey.indexOf("|");
        skey = hkey.substring(0,index1);
        infohash = hkey.substring(index1+1);

        String[] tempo2 =null;
        tempo2 = infohash.split("|");
        int index2 = infohash.indexOf("|");
        info = infohash.substring(0,index2);
        hashp = infohash.substring(index2+1);

        String[] tempor1 =null;
        tempor1 = info.split("-");
        int indexs1 = info.indexOf("-");
        algo = info.substring(0,indexs1);
        String kmp = info.substring(indexs1+1);

        String[] tempor2 =null;
        tempor2 = kmp.split("-");
        int indexs2 = kmp.indexOf("-");
        String keylength = kmp.substring(0,indexs2);
        String mp = kmp.substring(indexs2+1);

        String[] tempor3 =null;
        tempor3 = mp.split("-");
        int indexs3 = mp.indexOf("-");
        String mode = mp.substring(0,indexs3);
        String padding = mp.substring(indexs3+1);
    }
int keylengths = Integer.parseInt(keylength);

if (algo.equals("AES")){
    if (keylengths == 256){
        try{
            aesKey = new byte[32];
            aesKey = Base64.decodeBase64(skey.getBytes());
            key = new SecretKeySpec(aesKey, "AES");
            aesCipher = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
        } catch (Exception e){}
    }
    else if (keylengths == 192) {
        try{
            aesKey = new byte[24];
            aesKey = Base64.decodeBase64(skey.getBytes());
            key = new SecretKeySpec(aesKey, "AES");
            aesCipher = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
        } catch (Exception e){}
    }
    else if (keylengths == 128) {
        try{
            aesKey = new byte[16];
            aesKey = Base64.decodeBase64(skey.getBytes());
            key = new SecretKeySpec(aesKey, "AES");
            aesCipher = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
        } catch (Exception e){}
    } else
        System.out.println("Only support AES 256/192/128...");
}while AES
return hashp;
}

public void decrypt(File in, File out) throws IOException, InvalidKeyException {
    byte[] ivBytes = new byte[] {
        0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00,
        0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00};

    IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);
    try{
        aesCipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
    } catch (InvalidAlgorithmParameterException e){System.out.println(e);} 

    CipherInputStream is = new CipherInputStream(new FileInputStream(in),
    aesCipher);
    FileOutputStream os = new FileOutputStream(out);

    copy(is, os);
    is.close();
    os.close();
}

private void copy(InputStream is, OutputStream os) throws IOException {
    
}
int i;
byte[] b = new byte[1024];
while((i=is.read(b))!=-1) {
    os.write(b, 0, i);
}

public String processCiphertext(File fhashkey, File fctext )throws Exception{
    File fplain = new File("plaintext.txt");
    String hashp = null;

    try{
        hashp = loadKey(fhashkey, pubKey);
        decrypt(fctext, fplain);
    }catch(GeneralSecurityException ioe){ }
    catch(Exception ioe){ }
    return hashp;
}

}//class Code
Appendix B: Publications


3. Rossilawati Sulaiman, Xu Huang, and Dharmendra Sharma, "E-health Services with Secure Mobile Agent", Annual Conference on Communication Networks and Services Research, pp. 270-277, CNSR 2009, Los Alamitos, CA, USA

BIBLIOGRAPHY


Fibiková, L., & Vyskoc, J. (2001). Practical cryptography - the key size problem: PGP after years


FIPA-ACL. from http://www.fipa.org/repository/aclspecs.html


Office for Civil Rights.


Marco de, V., Gabriela, O. d. V., & Germinal, I. (1998). Internet security attacks at the basic levels (Vol. 32, pp. 4-15): ACM.


Miliard, M. (2010). Healthcare industry one of the most mobile


