Australian Journal of Basic and Applied Sciences, 6(9): 588-604, 2012
ISSN 1991-8178

Securing Online Communications using MLC and MAS in E- Health

[1]Rossilawati Sulaiman, [2]Dharmendra Sharma, [2]Wanli Ma, and [2]Dat Tran

[1]School of Computer Science, Faculty of Information Science and Technology, National University of Malaysia, 43600, Bangi, Selangor, Malaysia.
[2]Faculty of Information Science and Engineering, University of Canberra, Bruce, 2601, ACT, Australia.

Abstract: In this paper, we establish a new security mechanism for online communications among users, specifically in the e-health domain. E-health, as a medium for information exchange among health practitioners, often involves exchanging and sharing information, which should be treated with utmost confidentiality. The multilayer communication approach (MLC) is used to provide a flexible way for users to select the best security methods to secure their communications. We choose multi agent system (MAS) as a supporting tool to implement MLC, because of their characteristics such as autonomous, interactive, extendible and mobile to cater for the security processes. We develop an agent-based system called MAgSeM, as well as a traditional non agent-based system, using Java socket system and run both systems to simulate a two-way communication. We evaluate the security performance of the two systems, in terms of the execution time and the overhead costs of the security processes. We found out that although the MAgSeM-based system performed slower than the Socket-based system, it gives better control and flexibility on the security to the sender of the message as well as security automation, compared with the Socket-based system.

Key words: Multiagent system (MAS), multilayer security approach (MLC), cryptography protocols.

INTRODUCTION

Electronic health (or e-health) uses the Internet to enhance healthcare service deliveries, through online communications. Currently, communications in e-health involve applications that support online communication such as videoconferencing sessions, electronic mails, and web-based applications (Alpay *et al*., 2010;Chen, *et al*., 2001; Svenson, 2002). Communication among medical practitioners as well as medical practitioners and patients, often involve message exchanges related to health and medication issues, which obviously must occur in a private and secure environment.

However, the Internet has its own history of threats such as network attacks, information privacy/sensitivity breaches, and malicious software. Many concerns have been expressed regarding the security in the health domain (Tyler, 2001; Dearne, 2006; Liu *et al*, 2011). Current security technologies such as SSL/TLS, IPSec, SSH, or VPN have been robustly exercised, offering excellent security mechanisms to online communications. In reality, in order to use such technologies, one must configure the security setting (such as in SSL) to select appropriate ciphersuites for cryptography purposes for a particular communication. However, in this paper, we are interested to find 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, a one-to-one or one-to-many communications can be established using different security strengths simultaneously, without having to reconfigure the security setting. However, with the current security technologies, one will always have to reconfigure the ciphersuites every time one needs a much stronger or much weaker security.

In order to achieve this flexible type of security, we decided to use the multi-agent system (MAS) approach and the agents' characteristics to cater for security processes. There are many applications based on MAS to secure online communications such as found in (Juan, 2008; Nikooghadam and Zakerolhosseini, 2012). In our agent system, the agents are skilled with knowledge to handle security processes. Mobile agents are used to carry sensitive information as well as the agent's code from one host to another. Cryptographic protocols are used to secure the data as well as the agent's code.

MATERIALS AND METHODS

The Multilayer Communication Approach (MLC):

MLC is an approach that provides flexibility to the communication by classifying communications between different categories of users into five different layers based on requirements. Each layer is labelled from 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

Corresponding Author: Rossilawati Sulaiman, School of Computer Science, Faculty of Information Science and Technology, National University of Malaysia, 43600, Bangi, Selangor, Malaysia.
E-mail: ross@ftsm.ukm.my" <ross@ftsm.ukm.my

communication. To ease the understanding on MLC, we provide a scenario of the communication between actors in an e-health organization (such as a hospital) , which are divided into seven groups of communications:

1. Doctor ⇔ Doctor,
2. Doctor ⇔ Patient,
3. Doctor ⇔ Nurse;
4. Nurse ⇔ Patient,
5. Paramedic ⇔ System Coordinator (SC),
6. Social Worker (SW) ⇔ Doctor, Nurse;
7. System Administrator (SyA) ⇔ Doctor, Nurse, Patient, SW, SC, Paramedic

The '⇔' symbol indicates a two-way communication between the actors. There are communications involving wired communication and wireless communications. Wireless communications implies the use of lightweight devices in the communications. For example, a paramedic at an accident spot uploads important information about the patient(s) to the SC at the hospital so that SC could arrange available medical team when the patient(s) arrived at the hospital.

Different Security Strength for Different Layers:

MLC uses different kinds of security layers for each group of communication, based on the concept of information classification guideline, ISO 17799 (Sulaiman *et al*., 2011). There are five layers of communications in MLC as shown in Table 1, identified as Top Secret, Highly Confidential, Proprietary, Internal Use Only, and Public; which each shows the different level of data sensitivity, based on the communications among the actors in e-health.

Table 1: Five layers of communications in MLC.

Layer	Sensitivity of the data	Types of data	Users
1	Top Secret	Contains Extremely Sensitive information	Doctor⇔Doctor Doctor⇔Patient Doctor⇔Nurse Nurse⇔ Patient
2	Highly Confidential	Contains Highly Sensitive information	Paramedic ⇔ SC
3	Proprietary	Contains Medium sensitive information	Doctor⇔SW Nurse⇔ SW Patient⇔SW
4	Internal Use Only	Contains Low sensitive information	SyA ⇔ all users
5	Public	*Open channel:* No sensitive information	The public

With MLC, each layer is protected using different security protections. The MLC provides three types of security mechanisms, which are data security, channel security, as well as data and channel security. For data security, cryptography protocols such as encryption/decryption, hash function, and digital signature are used. Meanwhile, SSL is used for channel security. Each layer is protected with different security strengths to provide flexibility to the users to choose which layer is suitable for them. The *strengths* discuss here refer to the different key lengths used for symmetric encryption/decryption. The details about the cryptography protocols will be discussed in the Security Protocol Section. Table 2 shows the key lengths recommendation for each layer in MLC. Readers are referred to (Sulaiman *et al*., 2011) for further understanding of the key lengths.

Table 2: Key length recommendation for each layer in MLC.

MLC	Key lengths (in bit)
Layer 1 (Extremely Sensitive)	193 and longer
Layer 2 (Highly Sensitive)	Wired: 129-192
	Lightweight device: 112-192
Layer 3 (Medium Sensitive)	112-128
Layer 4 (Low Sensitive)	80-111

Communication Layer:

Communication Layer or *com_layer* is the implementation of MLC that refers to the five layers of communications. *com_layer* determines the security mechanisms and symmetric key lengths, which will be applied to information in a particular communication session. *com_layer* is chosen by assigning a default layer value (labelled as L0) to each user, based on the sensitivity of the data each user may carry, like the following:

1. Patient, Doctor, and Nurse : L0 = Layer 1
2. Paramedic and SC : L0 = Layer 2

3. SW : L0 = Layer 3
4. SyA : L0 = Layer 4.

Users that communicate Extremely Sensitive information is assign to smaller L0, while users that communicate Low Sensitive information will be assign a higher value. The rules to determine *com_layer* for a communication between a sender and a recipient are as follows:

1. If L0 for the sender and recipient are the same, then *com_layer* for that communication will be the recipient's L0 (e.g.: Patient⇔Doctor, both L0s are 1, therefore their *com_layer*: Layer 1).
2. If L0 for the sender is greater than the recipient's, then *com_layer* for that communication is the sender's L0 (e.g.: SW⇔Doctor, L0s are 3 and 1, therefore their *com_layer*: Layer 3).
3. If L0 for the sender is smaller than the recipient's, then *com_layer* for that communication is the recipient's L0 (e.g.: Nurse⇔SyA L0s are 1 and 4, therefore their *com_layer*: Layer 4).

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 the *com_layer*. The *com_layer* value is associated with the length of the symmetric key encryption algorithms such as proposed in Table 2.

Implementing MLC with Multiagent System:

We identify several agents' characteristics to be used with MLC. There are many characteristics such as the ability to coordinate and cooperate to achieve the overall system goal, autonomous, extensible, interactive, and mobile. MAS complies with the FIPA standard that provide specifications for agent communications and agent management, which are important to coordinate and cooperate with each other (Bellifemine *et al*., 2007).

An agent is autonomous (lesser, 1999), that is, it can have its own state and data to take initiative. When an agent is executed, it will execute its predefined behaviour without being invoked by other objects or other agents. 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. Autonomous gives an advantage to automate the security processes. A sender's agent collaborates with the agent from the recipient's to send a message and to determine what kind of encryption algorithm that should be used in the communication session.

Agent is extensible (Sycara, 1998), in a sense that it could instantiate a new agent with different skills to perform tasks in order to achieve the overall system's goal. As a result, it allows an agent to be added or 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.

Lastly, an agent can be mobile (Braun and Rossak, 2004; Huhns and Singh, 1998). It can be launched to carry a secure message to the recipient's side and handles the security processes there, without the user involvement. Our agent system is called MAgSeM, which consist of 10 agents.

Agents in MagSeM:

Figure 1 shows the architecture of agents in MAgSeM. The agents are identified as Interface Agent (IA), which interacts and obtains data from the user, Data Organizer Agent (DOA), organizes the data received from the user such as the message and the recipient's address, Multi-tasking Agent (MTA) that makes a request to send a message to other users and keeps track of undelivered messages, Crypto Agent (cA), to provide all necessary information and parameters for the security processes, SetUp Agent (SUA), which applies cryptography protocols, Mobile Agent (MA) that carries secured data to the Recipient's platform, Communication Listener Agent (CLA) to listen to any incoming request, and Receiver Agent (RA) that provides verification service to the agents that arrive at the platform. There are also server Agent (SvA) resides at the server's side to manage the authentication process and other requests from agents, and lastly Decrypt Agent (DA) to perform the decryption process at the recipient's side.

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 shown as <*Algorithm, lengths, mode, padding*> where, *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. Any known symmetric encryption algorithms such as (AES, DES, Twofish, blowfish, TDES) can be put into the MLC specification.

The Security Protocols:

Consider a communication between Sender Agent (SUA) and Recipient Agent (RA), which uses a mobile agent (MA) to carry the message from the sender's side to the recipient's side. MAgSeM focuses on a control mechanism on how a sender can securely transfer data to a recipient while *maintaining control* over the data. This can be describe as (1) if the message carried by MA, is intercepted by an attacker, the attacker still cannot

claim the plaintext, and (2) RA does not need to know the details on how to recover the plaintext. One way for the sender to maintain 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. A symmetric key, *K* is used to encrypt the plaintext. This key and the information about the key (stored in the MLC specification), are kept with SUA until it knows that MA, which has moved to the recipient's host needs it.

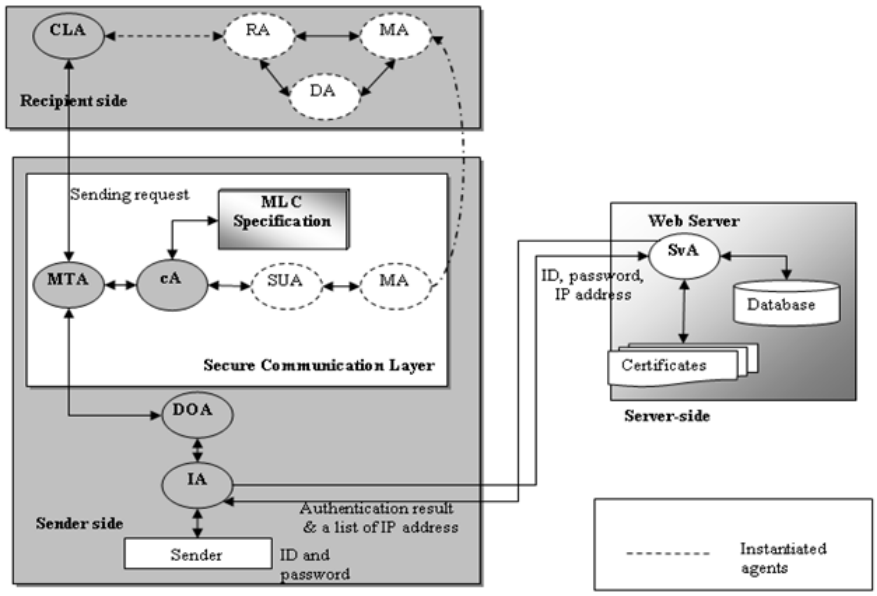

Fig. 1: Agent architecture in MagSeM.

A *token*, which is an encrypted random number, is carried by MA to the recipient's host. The token is sent back to SUA to let it knows that MA 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. In MAgSeM, two symmetric keys (*K1* and *K2*) are used in the security processes. An agent's code, which has the functionality to decrypt a ciphertext, is called as *Cd*, is migrated with MA to the recipient's side. When executed, this code will become DA. 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*, Hash of *P*: *H(P)*
f. Ciphertext: *C*
g. Signature: *S*
h. Agent's code: *Cd*
i. Hash of *Cd*: *H(Cd)*
j. A random number *Rand*, Token: *T*
k. The information extracted from the MLC Specification, *mlc*

Steps taken by SUA:

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 (*Ks, Kp*). After *T* is received, *Ks* is used to encrypt the information that is kept for decryption processes. The corresponding *Kp* will be embedded in *Cd* and sent to the recipient's host to be used for decryption.
7. Take *Cd* and create a .jar file
8. Signs the .jar file (*Cd*) with *privKs* to produce a signature, *S*, which is used to verify that *Cd* is from the sender.

$$S = E(Cd)privKs$$

9. Encrypts *Cd, S,* and *T* with *K2* to produce *Ciphercode*.

$$Ciphercode = E(Cd, S, T)K2$$

10. To allow only Receiver Agent to retrieve *K2,* it is encrypted with *pubKr* together with *H(Cd)* to produce *Cipherkey*.

$$Cipherkey = E(K2, H(Cd))pubKr$$

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

11. Saves *C*, *Ciphercode*, and *Cipherkey* in a file. Establishes SSL connection if necessary for channel security.
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), K1*, and *mlc* .

$$hashKey = E(K1,mlc,H(P))Ks$$

(Kp, Ks) pair is generated once per communication session. These keys will be removed once the session has finished, avoiding any third party from using Kp (which can be retrieved from the Recipient's host) in the next communication sessions. The signature S is 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).

Steps taken by MA:

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

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

Steps taken by RA:

RA will be in charge of communicating with MA and DA 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)*

$$D(Cipherkey)privKr = K2, H(Cd)$$

4. Use *K2* to decrypt *Ciphercode* to obtain *T, S* and *Cd*.

$$D(Ciphercode)K2= T, S, Cd$$

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, and execute *Cd* (which later be called DA). 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.

Steps taken by DA:

1. Once executed, DA 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.

Communication Setup:

We develop two systems namely the MAgSeM-based system and the Socket-based system, to simulate the two-way communication. The Socket-based system implementation is based on the Java Socket programming (Sun-Java, 1995), 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, such as depicted in Figure 2.

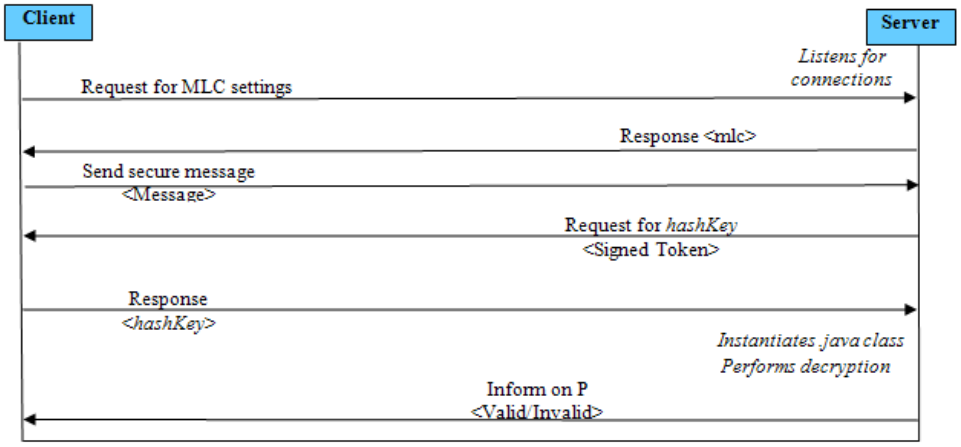

Fig. 2: Communication protocol for Socket-based system.

The Socket Based System:

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 with modifications like the following:

a. The client applies the same security protocols such as performed by the SUA 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, 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. 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.

The Magsem-Based System:

Consists of agents that reside in the sender's and recipient's hosts, which include parts of the whole communication among the agents: MTA, CLA, cA, SUA, MA, DA and RA. The communications are presented in Figure 3, for the communication protocol. Messages among agents are described in "*performative: <msg1|msg2|msg$_n$>*" format.

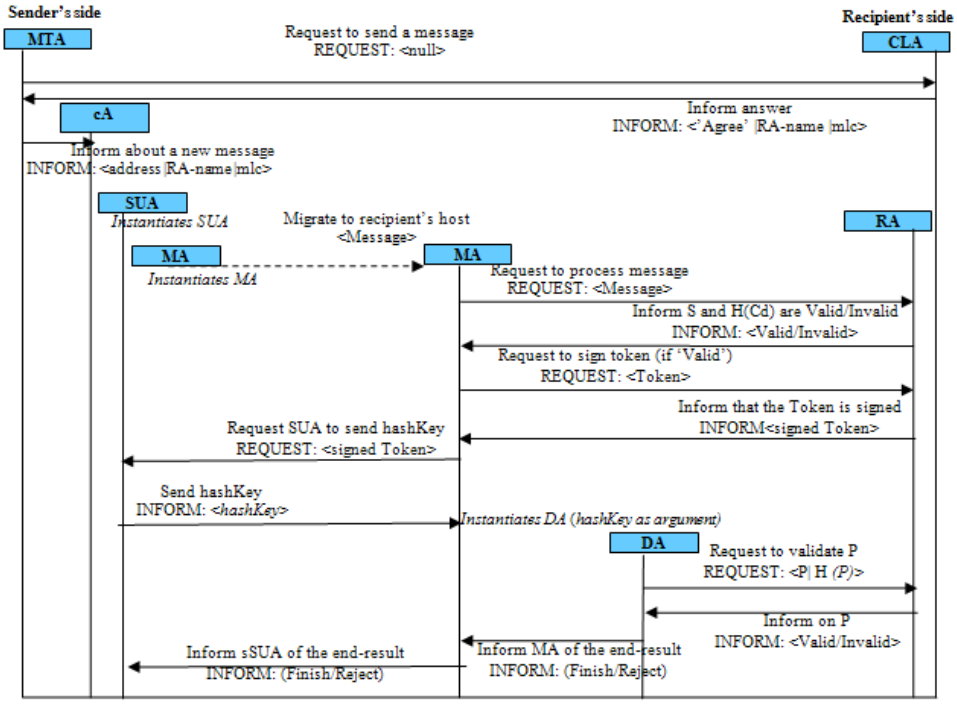

Fig. 3: Communication protocol for MAgSeM-based system.

Consider that MTA has accepted an '*Agree'* message from 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 sender (e.g. patient) and the recipient (e.g. doctor) based on their L_0s (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.

Result:

Experimental Setup:

In this research, the communication performance in each layer in the MagSeM-based system is measured, and compared with the Socket-based system, in term 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. We created a controlled environment using two PCs connected to each other in a LAN. Each computer was equipped with Windows XP professional, Pentium IV, 3 GHz CPU, and 1 GB RAM.

Table 3: Experiment setup for MAgSeM and Socket-based systems

No	Security Mechanisms	Algorithm for K1 and K2
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	-

Four MLC security settings and one with no security were used, as shown in Table 3. 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.

Time Measurement:

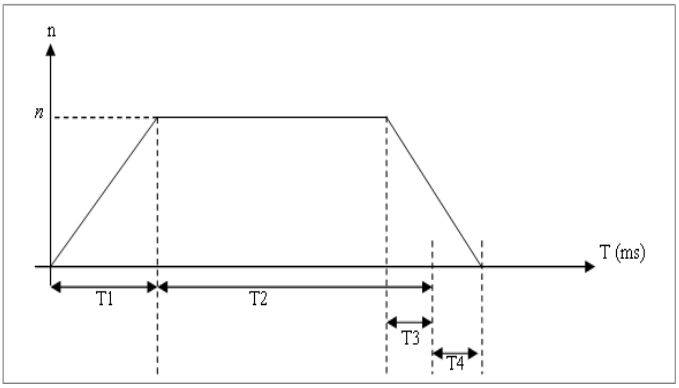

Fig. 4: Time intervals to complete a communication.

Our system is built with Java, and therefore, we use the Java method of System.currentTimeMillis(), which returns current time in 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). We identify four different measurements to complete the whole communication between sender and recipient, as shown in Figure 4. *n* is the average time taken for the agent, and T is the time to complete a communication in milliseconds.

1. T1 is the time taken for processing the data at the sender side, which is to generate Ciphertext, Cipherkey, and Ciphercode.
2. 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.
3. 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.
4. T4 is the time taken starting from receiving hashKey at the recipient's side, until the plaintext is discovered and validated.

TransacT and Transfer Time (TT):

TransactT is a time to complete a circular message exchange, calculated as:

TransacT = T1 + T2.

By calculating TransacT, we are also able to examine the transfer time, 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, T4 is not included in the experiment. 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.

Experimental Result:

In the experiments, we observe the following execution times:

1. Processing 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. Processing Time at the Recipient's side, T3, and
4. Transaction Time, TransacT; a time taken to complete a circular message exchange.

Processing Time (T1):

As explained before, T1 is the time taken to generate *Ciphertext*, *Ciphercode*, and *CipherKey*. Table 4 and Table 5 show the result of generating *Ciphertext* for the MAgSeM-based and Socket-based systems respectively. 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 MAgSeM-based system.

Table 4: Time measurements for Ciphertext (MAgSeM-based).

	1M	3M	5M	7M	9M
AES 256 + SSL	84.40	221.40	352.13	496.93	631.83
AES 192	83.80	218.73	339.50	465.63	591.80
AES 128	77.03	205.23	314.03	429.57	543.77

Table 5: Time measurements for Ciphertext (Socket-based).

	1M	3M	5M	7M	9M
AES 256+ SSL	82.20	219.20	354.67	485.47	615.20
AES 192	80.73	208.30	327.10	463.07	583.30
AES 128	70.80	191.10	301.03	427.73	540.57

For *Cipherkey* and *Ciphercode* generation, the results are shown in Figure 5 for the MAgSeM-based system and Figure 6 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 and Figure 8 respectively. Similar pattern of results were generated for both systems, where the processing time increased with increased plaintext sizes.

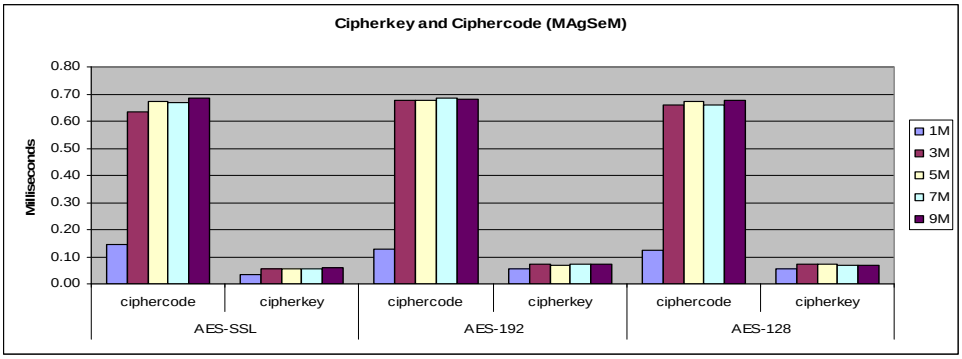

Fig. 5: Generating Cipherkey and Ciphercode for MAgSeM-based system.

The MAgSeM-based system took a less time to complete T1 compared to the Socket-based system. Table 6 and Table 7 show the value of T1 for both systems.

Table 6: T1 for MAgSeM-based system.

	1M	3M	5M	7M	9M
AES 256 + SSL	845.23	1408.80	1866.06	2355.77	3056.33
AES 192	1000.46	1520.43	1958.30	2417.767	3158.43
AES 128	875.63	1506.76	1946.33	2347.367	3131.90

Table 7: T1 for Socket-based system.

	1M	3M	5M	7M	9M
AES 256 + SSL	1057.76	1598.40	2137.93	2680.80	3180.76
AES 192	1558.33	2078.63	2610.96	3304.23	3787.56
AES 128	1557.8	2065.10	2592.23	3270.80	3742.66

We calculated the percentage increase/decrease of MAgSeM-based system against the Socket-based system as:

Percentage Increase/Decrease = $((Socket_{value} – MAgSeM_{value}) /MAgSeM_{value}) * 100$

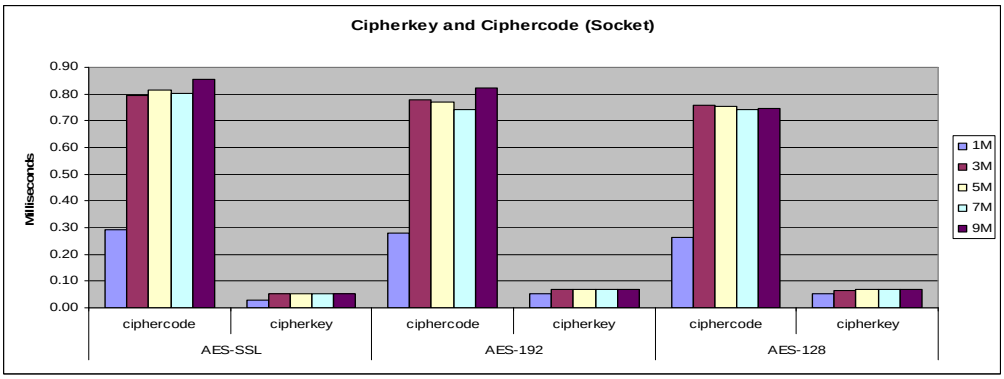

Fig. 6: Generating Cipherkey and Ciphercode for Socket-based system.

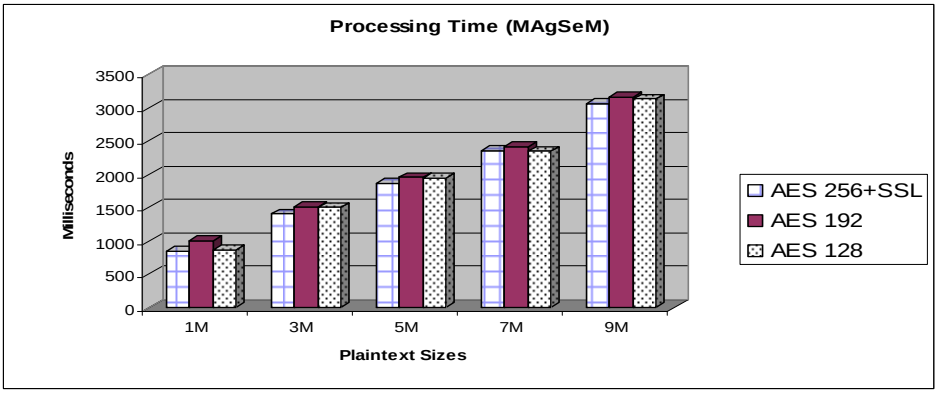

Fig. 7: Comparison for T1 for MAgSeM-based system.

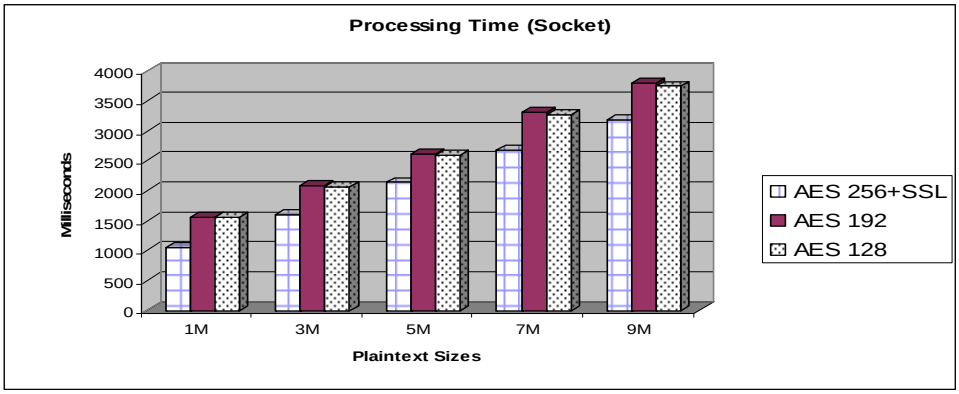

Fig. 8: Comparison for T1 for Socket-based system.

As observed in Table 8, 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.

Table 8: Percentage increased of the MAgSeM-based system for T1.

	1M	3M	5M	7M	9M
AES 56 + SSL	25.15	13.46	14.57	13.80	4.07
AES 192	55.76	36.71	33.33	36.66	19.92
AES 128	77.91	37.06	33.19	39.34	19.50

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 9 and Figure 10 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.

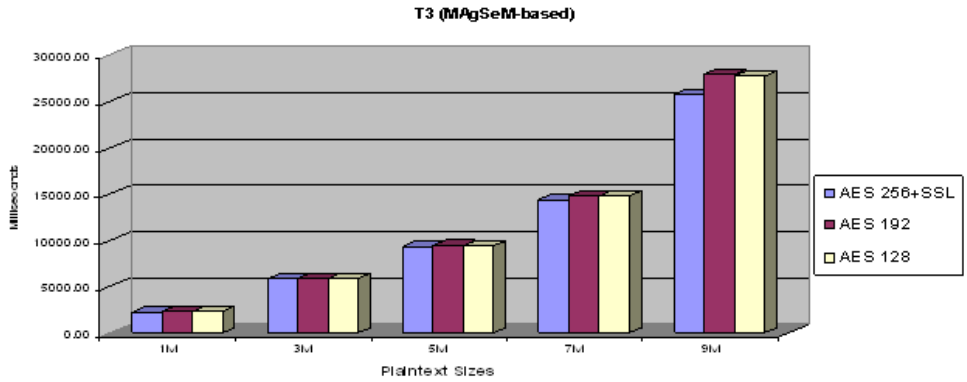

Fig. 9: Comparison for T3 for MAgSeM-based system.

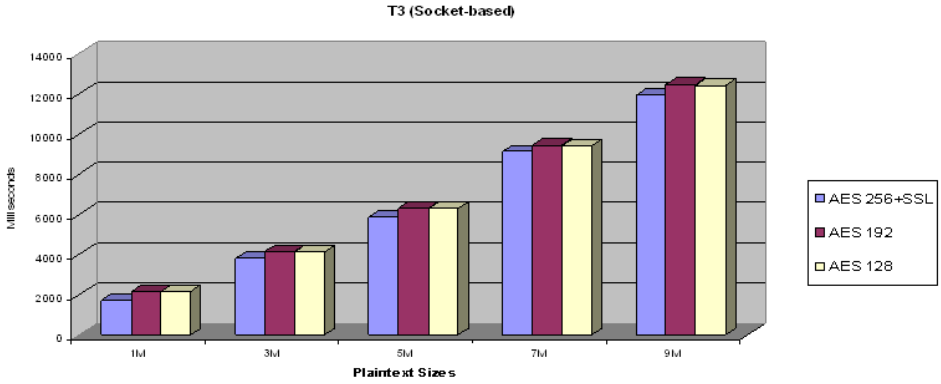

Fig. 10: Comparison for T3 for Socket-based system.

Table 11 shows the percentage decreased of the MAgSeM-based system, which were calculated from T3 values of both systems (given in Table 9 for MAgSeM-based and Table 10 for Socket-based systems). 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 9: T3 for MAgSeM-based system.

	1M	3M	5M	7M	9M
AES 256 + SSL	2220.37	5829.60	9385.87	14382.30	25787.43
AES 192	2339.53	5913.10	9432.40	14825.00	27941.13
AES 128	2334.77	5901.57	9414.73	14815.07	27800.07

Table 10: T3 for Socket-based system.

	1M	3M	5M	7M	9M
AES 256 + SSL	1705.83	3785.47	5862.00	9111.03	11917.27
AES 192	2147.90	4142.23	6288.57	9456.90	12445.87
AES 128	2146.83	4133.77	6283.90	9405.17	12400.03

Table 11: Percentage decreased of the MAgSeM-based system for T3.

	1M	3M	5M	7M	9M
AES 256 + SSL	-23.17	-35.06	-37.54	-36.65	-53.79
AES 192	-8.19	-29.95	-33.33	-36.21	-55.46
AES 128	-8.05	-29.95	-33.25	-36.52	-55.40

Transfer Time (TT):

TT represents the time taken for 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 11 and Figure 12 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 (Juan, 2008) to transfer large data sizes utilized multiple ACL messages, which added to the total transfer time.

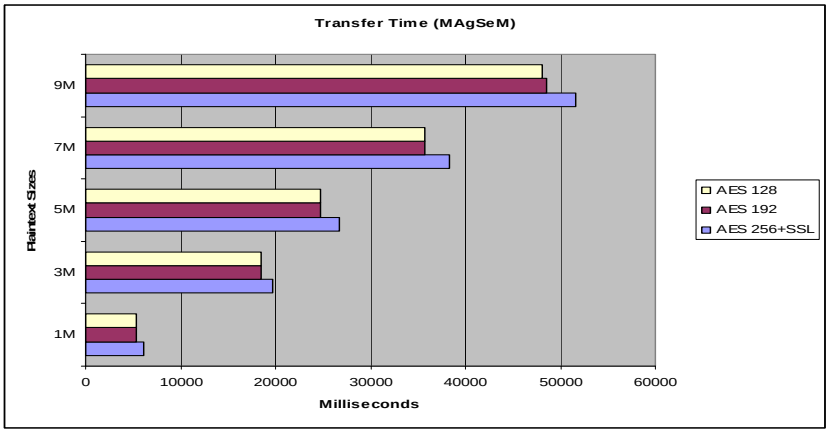

Fig. 11: Comparison for TT for MAgSeM-based system.

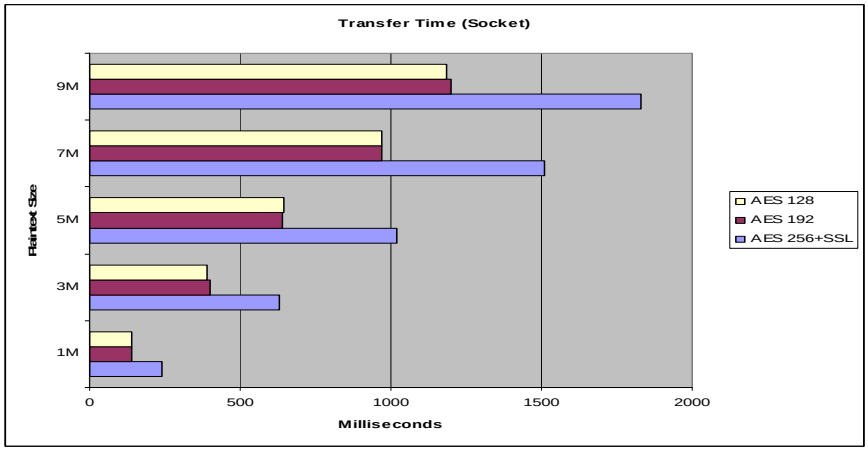

Fig. 12: Comparison for TT for Socket-based system.

The values of TT for both systems are shown in Table 12 for the MAgSeM-based system and Table 13 for the Socket-based system. We found a decreased of percentage for the MAgSeM-based system (shown in Table 14), where all layers performed nearly 100% slower than the Socket-based system.

Table 12: TT for MAgSeM-based system.

	1M	3M	5M	7M	9M
AES 256 +SSL	6084.33	19759.50	26646.47	38298.97	51665.73
AES 192	5380.30	18496.33	24695.70	35712.93	48562.00
AES 128	5379.33	18503.27	24692.60	35701.07	48164.03

Table 13: TT for Socket-based system.

	1M	3M	5M	7M	9M
AES 256 +SSL	238.93	630.17	1019.73	1510.90	1830.67
AES 192	141.57	401.63	638.07	969.10	1199.03
AES 128	140.73	391.67	646.73	971.97	1186.43

Table 14: Percentage decreased of the MAgSeM-based system for TT.

	1M	3M	5M	7M	9M
AES 256 +SSL	-96.07	-96.81	-96.17	-96.05	-96.46
AES 192	-97.37	-97.83	-97.42	-97.29	-97.53
AES 128	-97.38	-97.88	-97.38	-97.28	-97.54

Transaction Time (TransacT):

The results for TransacT for the MAgSeM-based system and the Socket-based system are shown in Figure 13 and Figure 14 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.

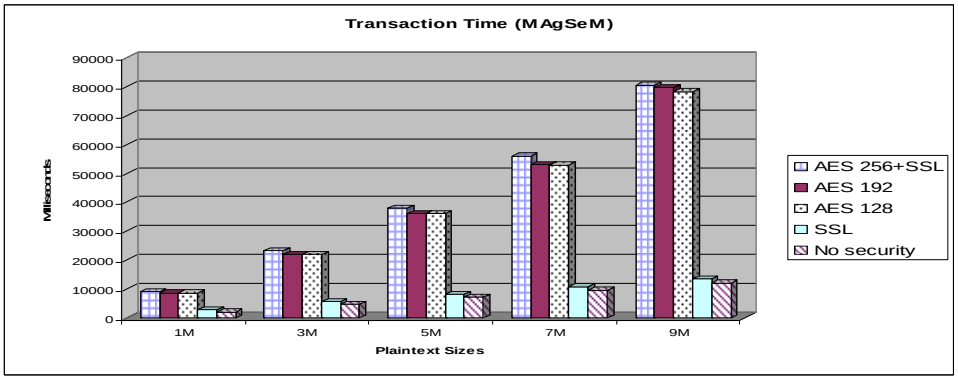

Fig. 13: Execution time for TransacT for MAgSeM-based.

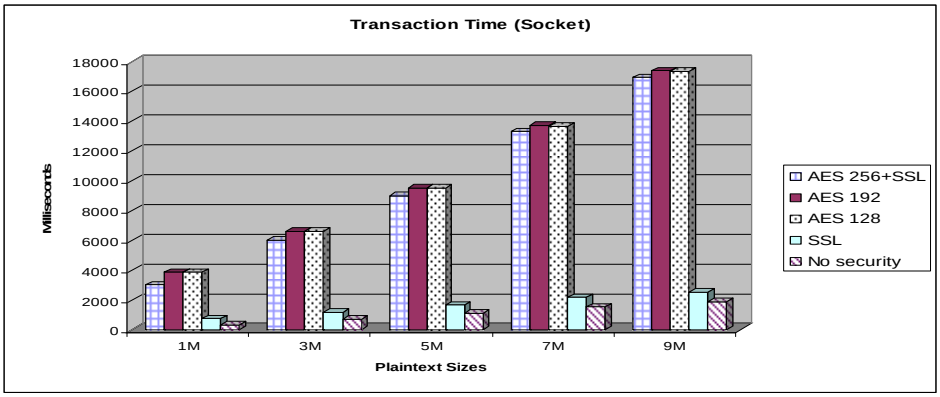

Fig. 14: Execution time for TransacT for Socket-based.

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. 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. Moreover, the size of the plaintext transferred by the mobile agent did not change. 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 17, which were obtained from TransacT values of both systems (as presented in Table 15 and 16). 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 15: TransacT for MAgSeM-based system.

	1M	3M	5M	7M	9M
AES 256+SSL	9149.93	23155.27	37898.40	56037.03	80509.50
AES 192	8720.30	21986.57	36068.73	52955.70	79661.57
AES 128	8589.73	21972.50	36071.33	52863.50	78238.00
SSL	2829.30	5667.10	8220.30	10782.78	13533.00
No security	2118.70	4754.80	7234.30	9609.40	12118.70

Table 16: TransacT for Socket-based system.

	1M	3M	5M	7M	9M
AES 256+SSL	3002.53	6014.033	9019.67	13302.73	16928.70
AES 192	3847.80	6614.03	9537.60	13730.23	17386.63
AES 128	3845.37	6599.00	9522.87	13647.93	17374.97
SSL	746.80	1170.30	1642.30	2176.80	2535.80
No security	337.60	714.20	1081.60	1518.60	1873.50

Table 17: Percentage decreased of the MAgSeM-based system for TransacT.

	1M	3M	5M	7M	9M
AES 256+SSL	-67.19	-74.03	-76.20	-76.26	-78.97
AES 192	-55.88	-69.92	-73.56	-74.07	-78.17
AES 128	-55.23	-69.97	-73.60	-74.18	-77.79
SSL	-73.60	-79.35	-80.02	-79.81	-81.26
No security	-84.07	-84.98	-85.05	-84.20	-84.54

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 18 and Table 19. The tables summarise the results for TransacT for the MAgSeM-based and Socket-based systems respectively.

Table 18: TransacT values in ms for 7 Mb (MAgSeM-based).

Security Setting	7M
Layer 1: AES 256+SSL	56037.03
Layer 2: AES 192	52955.70
Layer 3: AES 128	52863.50
Layer 4: SSL	10782.78
Layer 5: No security	9609.40

Table 19: TransacT values in ms for 7 Mb (Socket-based).

Security Setting	7M
Layer 1: AES 256+SSL	13302.73
Layer 2: AES 192	13730.23
Layer 3: AES 128	13647.93
Layer 4: SSL	2176.80
Layer 5: No security	1518.60

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:

$$PSO = ((TransacT_{n} - TransacT_{ns})/TransacT_{ns}) * 100,$$

Where $TransacT_n$ is the TransacT for Layer-n, and $TransacT_{ns}$ is the TransacT for No security communication. Table 20 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 20: PSO for 7Mb communications (MAgSeM-based).

Security setting	Percentage
Layer 1	483.15
Layer 2	451.08
Layer 3	450.12
SSL only	12.21

Table 21: PSO for 7Mb communications (Socket-based).

Security Setting	Percentage
Layer 1	775.99
Layer 2	804.14
Layer 3	798.72
SSL only	43.34

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 21). 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 15). This is in contrast with the Socket-based system that only took 1518.60 ms (reading from Table 16) 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.

Discussion:

We have presented the evaluation of security performances, in terms of end to end execution times for MAgSeM-based and Socket-based systems. The results from the previous sections lead to the following conclusions:

1. The *Ciphertext* generations were faster using shorter key lengths. The *Cipherkey* generations were faster than the generation of *Ciphercode*.
2. 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.
3. For the processing time at the recipient's side, Layer 1 performed faster than the other layer.
4. Layer 1 has the longest transfer time because of the channel security overhead. Layer 3 performed slightly faster than Layer 2.
5. 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.
6. 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.

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.

From the design perspective, we argue that the MAgSeM-based system has advantages to provide a much better control on security compared to the traditional non-agent based system, by using 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, by using *com_layer*, which determines the security mechanisms stored in the MLC specifications. In contrast with the existing non-agent based system, the technologies only provides one type of security mechanism for every communication, such can be found in the

SSL, IPSec, or VPN. These security mechanisms require reconfiguration in the set up process in order to change the security level. On the other hand, MAgSeM-based system can automate the security processes in such a way that users do not need reconfigurations.

In MAgSeM, MA can be instantiated and migrated to carry a secure message to the recipient's 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 (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, SUA can hold *K1* and the information about it 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. 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 the Socket-based system implementation, we have identified several. Firstly, communications that are similar to *agent-to-agent communications* are not supported. In the case of the agent-to-agent communications, 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. Reports sends among these agents can be expected milestones 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.

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.

In addition, the server is burdened by the implementation of the Java .class object as in Figure 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.

Conclusion:

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 that MagSeM-based system provides (1) flexible security mechanisms and (2) security automation. For our future work, we will investigate in details on the security for wireless communications in e-health.

REFERENCES

Bellifemine, F.L., G. Caire and D. Greenwood, 2007. Developing Multi-Agent Systems with JADE: Wiley.

Braun, P. and W. Rossak, 2004. Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit: Morgan Kaufmann Publishers Inc.

Chen, T.L., Y.F. Chung, F.Y. Lin, 2012. Deployment of Secure Mobile Agents for Medical Information Systems. J Med Syst., 36(4): 2493-503.

Chen, Z., X. Yu and D. Feng, 2001. Telemedicine system over the internet. In the ACM International Conference Proceeding Series, Selected papers from the Pan-Sydney workshop on Visualisation, pp: 113-118.

Dearne, K., 2006. Unencrypted medical records email is like using postcard, says doctor. The Australian, IT Today.

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC135526/pdf/1476-3591-1-5.pdf

Huhns, M.N. and M.P. Singh, 1998. Readings in Agents. San Francisco, Calif.: Morgan Kaufmann Publishers.

Juan, J.C., 2008. Efficient Mobility and Interoperability of Software Agents., Universitat Autònoma devBarcelona,.

Laurence, L., Alpay, Olivier Blanson Henkemans, Wilma Otten, Ton A.J.M. Rövekamp, and Adrie C.M.D., 2010. Telemedicine and e-Health., 16(7): 787-791.

Lesser, V.R., 1999. Cooperative Multiagent Systems: A Personal View of the State of the Art. IEEE Trans. On Knowledge and Data Engineering., 11(1): 133-142.

Liu, C.H., Y.F. Chung, T.S. Chen, S.D. Wang, 2010. The Enhancement Of Security In Healthcare Information Systems. J Med Syst, 36(3):1673-88.

Nikooghadam, M., A. Zakerolhosseini, 2012. Secure Communication of Medical Information Using Mobile Agents. Journal of Medical Systems, 36(6): 3839-3850.

Roth, V., 2002. On the Robustness of Some Cryptographic Protocols for Mobile Agent Protection. In the Proceedings of the 5th International Conference on Mobile Agents., pp: 1-14.

Sulaiman, R., D. Sharma, W. Ma, D. Tran, 2011. A New Security Model Using Multilayer Approach for E-Health Services. Journal of Computer Science, 7(11): 1691-1703.

Sun-Java., 1995. A Java Tutorial. Lesson: All About Sockets. http://java.sun.com/docs/books/tutorial/networking/sockets/index.html.

Sun-Microsystems., 2003. System (Java 2 Platform SE v1.4.2) http://72.5.124.55/j2se/1.4.2/docs/api/java/lang/System.html.

Svensson, P.-G., 2002. eHealth applications in health care management, EHealth International

Sycara, K., 1998. MultiAgent Systems. AI Magazine, 19: 79-92.

Tyler, J.L., 2001. The Healthcare Information Technology Context: A Framework for Viewing Legal Aspects of Telemedicine and Teleradiology. In the 34th Annual Hawaii International Conference on System Sciences (HICSS-34).