

*Mobile Trade Agent Security*

*Submitted by*

*Shaima Hameed Al-Khalifa*

*Supervisor*

*Prof. Mohammad Al- Haj Hassan*

*A Thesis Submitted in Partial Fulfillment of the Requirements for  
the Degree of Master of Science in Computer Information System*

**Faculty of Information Technology**

**Department of Computer Information System**

**Amman - Jordan**

**August 2011**

## جامعة الشرق الأوسط

### نموذج تفويض

انا شيماء حميد آل خليفه، أفوض جامعة الشرق الأوسط بتزويد نسخ من رسالتي ورقيا  
والكترونيا للمكتبات، أو المنظمات، أو الهيئات والمؤسسات المعنية بالأبحاث والدراسات  
العلمية عند طلبها.

الاسم: شيماء حميد سعيد

التاريخ: 2011 / 8 / 13

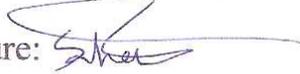
التوقيع: 

**Middle East University**  
**Authorization Form**

**I, Shaima Hameed Al-Khalifa,** authorize the Middle East University to supply copies of my Thesis to libraries or establishments or individuals on request.

Name: Shaima Hameed

Date: 2011 / 8 / 13

Signature: 

**Middle East University**  
**Examination Committee Decision**

This is to certify the thesis entitled "*Mobile Trade Agent security*" was successfully defended and approved on August 13<sup>th</sup> 2011.

**Examination Committee Members**

**Signature**

Prof. Mohammad Al- Haj Hassan ,  
Professor, Department of Computer Science  
(Middle East University)



Dr. Hazim A. Farhan ,  
Assistant Professor, Department of Computer Science  
(Middle East University)



Dr. Abdul Fattah Al- Tamimi  
Assistant Professor, Department of Computer Science  
(Alzaytoonah University)



## **Acknowledgments**

*This thesis owes a great deal to my supervisor, Professor Mohammad Al- Haj Hassan. I sincerely thank him for his guidance, wisdom, and encouragement. And I am grateful for his suggestions and comments during every step of the project.*

*I would further like to acknowledge all of the Information Technology faculty members at the Middle East University, for helping and encouraging my efforts especially at the beginning of the thesis.*

*Above all, I would like to especially thank my parents for supporting me during the time I was writing this thesis. Without them nothing of this thesis would have been possible.*

## **Dedication**

*I want to thank Allah whom by his willing I was able to write this thesis. This is dedicated to my father, for his support and care. To my mother, without her prayers and encouragement I couldn't finish my thesis. And also to everyone who stands behind us all the time, especially my brothers, friends and colleagues.*

<b>Table of Contents</b>	<b>Page</b>
List of Tables .....	VIII
List of Figures .....	IX
List of Abbreviations .....	X
Abstract .....	XI
Abstract in Arabic .....	XII
<b><i>Chapter One: Introduction to Mobile Trade Agent System</i></b>	
1.1 Introduction .....	2
1.1.1 Agent Types .....	4
1.1.2 Available Mobile Agents Examples .....	5
1.1.3 Security threats of mobile agents .....	6
1.2 Problem Definition .....	7
1.3 Thesis Objectives .....	9
1.4 Thesis Motivation .....	10
1.5 Thesis Contribution .....	10
1.6 Thesis Significance .....	11
1.7 Thesis Organization .....	11
<b><i>Chapter Two: Literature Review and Related work of Mobile Trade Agent System</i></b>	
2.1 Introduction .....	13
2.2 Literature Survey Related Mobile Agents Secure Migration .....	13
2.3 Protecting Mobile Agents using public key technique .....	15

2.4 Related studies on Protecting Mobile Agents .....	18
2.5 The Public-Key Cryptography Concept .....	22
2.5.1 Advantages and Disadvantages of Public-Key Cryptography .....	22
2.5.2 Introduction to RSA and Elgamal algorithms .....	24
<b><i>Chapter Three: Mobile Trade Agent Architecture and the Proposed Model</i></b>	
3.1 Introduction .....	26
3.2 Mobile Trade Agent Schema .....	26
3.3 The Proposed Model .....	29
3.4 The Algorithms Used in The System .....	32
3.4.1 Generate keys .....	32
3.4.1.1 RSA Public and Private Keys Generation .....	33
3.4.1.2 Elgamal Public and Private Keys Generation .....	35
3.4.2 Encryption Phase .....	36
3.4.3 Decryption phase .....	38
3.5 Worked Example .....	40
3.6 System Application .....	44
3.7 System Requirements .....	49
<b><i>Chapter Four: Implementation and Evaluation of the Proposed Model</i></b>	
4.1 Implementation of the Proposed Model .....	46
4.1.1 Authentication .....	47
4.1.2 Message Encryption .....	49
4.1.3 Message Decryption .....	50
4.1.4 The Application's Methods .....	50

4.2 Evaluation of the Proposed Model	.....	54
4.1.5 The Size of key	.....	58
<b><i>Chapter Five: Conclusion and Future Work</i></b>		
5.1 Conclusion	.....	60
5.2 Future Work	.....	61
References	.....	62
Appendix	.....	64

## List of Tables

Table 1 : Available Mobile Agents Examples .....	5
Table 2: RSA key size and the predicated time to be breakable in it .....	24
Table 3: The execution time of RSA & Elgamal algorithm.....	56
Table 4: The execution time of prime numbers .....	56

## List of Figures

Figure 1.1 : Shopping Agent, sent out to find best airfare .....	7
Figure 2.1 : Agent migration using a central authority (trusted third party) .....	15
Figure 2.2: The MTA Schema .....	18
Figure 2.4: Public key cryptograph .....	22
Figure 3.1: Mobile Trade Agent Architecture .....	28
Figure 3.2 The Proposed Mobile Agent System Model .....	30
Figure 3.3: RSA & Elgamal keys generation scheme.....	33
Figure 3.4: Two Phase Encryption .....	37
Figure 3.5: Two Phase Decryption .....	39
Figure 4.1 System Interface .....	46
Figure 4.2: Shows the user must enter the authentication key .....	47
Figure 4.3 The System Flow Chart .....	48
Figure 4.4: Message Encryption .....	49
Figure 4.5: Message Decryption .....	50
Figure 4.6: The execution time chart of RSA & Elgamal .....	56
Figure 4.7: The execution time chart of RSA & Elgamal (together).....	56
Figure 4.8: The execution time chart of prime numbers .....	58

## List of Abbreviations

<b>Abbreviate</b>	<b>Meaning</b>
AD	Agent Depositor y
AS	Authorization Server
FIPA	Foundation of Intelligent Physical Agents
FnC	Function Composition technique
DES	Data Encryption Standard
HES	Homomorphic Encryption Scheme
MARISM	Architecture for Mobile Agents with Recursive Itinerary and Secure Migration
MASIF	Mobile Agent Systems Interoperability Facilities Agents
MTA	Mobile Trade Agent
PKI	Public Key Infrastructure
RSA	Rivest, Shamir and Adleman
SNMP	Simple Network Management Protocol
SIAS	Shopping Information Agent System
TTP	Trusted Third Party

## **ABSTARCT**

### **Mobile Trade Agent security**

**By**

**Shaima Hameed Al-Khalifa**

**Supervised by**

**Prof. Mohammad Al- Haj Hassan**

Recently, e-commerce had been widely spread, and anyone can buy or sell online a certain product or service anywhere. As a result, this increased the need to find the best product with the best price. All of these led to use mobile agent trade systems to perform this task. Such systems should have the capability to act in behalf of the user and roam sites to find the list of goods or products with best price.

Unfortunately, a mobile agent system has some disadvantages, like being attacked by malicious hosts who try to steal and sabotage the mobile agent's data (the client credit card number for example).

Mobile agent system can be more successful and frequently used, if the security problems had been solved. The security is the major concern for Mobile agent system especially when the money is involved. Thus, we need a protection mechanism to ensure the integrity and safety of agent information.

In this thesis, an approach for protecting mobile agent had been designed, and a two-phase encryption protection application had been built using two cryptography algorithms: RSA & Elgamal. The proposed approach has been implemented and tested.

## المخلص

### أمن الوكيل التجاري الجوال

إعداد

شيماء حميد آل خليفه

إشراف

أ.د. محمد الحاج حسن

لقد انتشرت التجارة الألكترونية في الآونة الأخيرة، وأصبح بإمكان أي شخص أن يبيع ويشترى منتجاً ما أو يحصل على خدمة ما عن طريق الإنترنت، وهذا أدى إلى إزدياد الحاجة لإيجاد السلعة الأفضل و بأفضل سعر.

كل هذا قاد إلى إستخدام أنظمة الوكيل التجاري الجوال لأداء هذه المهمة، ولهذه الأنظمة القدرة على التصرف بالنيابة عن المستخدم، وتصفح صفحات الإنترنت من أجل ايجاد قائمة بالمنتجات بأفضل الأسعار.

إلا أن لأنظمة الوكيل التجاري بعض المضار، مثل تعرضها للهجوم من قبل المضيف، الذي يحاول تخريب وسرقة بيانات الوكيل التجاري (مثلا رقم بطاقة الائتمان).

إن أنظمة الوكيل التجاري يمكن أن تصبح أكثر نجاحاً وأكثر إستخداما اذا تم حل المشاكل المتعلقة بالأمنية، حيث تعد الأمنية الإهتمام الرئيسي لأنظمة الوكيل التجاري خاصة عندما يكون المال جزءاً من العملية، لذلك نحن بحاجة إلى آلية حماية من أجل ألتأكد من سلامة وأمن بيانات الوكيل التجاري .

في هذه الرسالة، تم تصميم طريقة لحماية الوكيل التجاري، وتم بناء تطبيق ذي مرحلتين لكل من التشفير و فك التشفير بإستخدام خورزميتي RSA والجمل، وقد تم برمجة وتنفيذ الطريقة المطروحة وإختبارها.

# *Chapter*

# *One*

# Chapter One

## Introduction to Mobile Trade Agent System

### 1.1 Introduction

The Inter-networked environment, such as the Internet, has made electronic commerce transactions more available than before, which also cause the increase of information about goods or services on the Internet.

The amount of information that is available on the Internet is much large that it becomes near impossible for humans to visit each site on the internet, analyze this information and choose the best merchandise to trade.

Therefore, there is a need for a trade agent that can roam sites, evaluate and decide where it is best to buy or sell goods on behalf of the user (Aqel, Aboud & Ahmed ,2007) .

Agents are independent pieces of software capable of acting autonomously in response to input from their environment. Agents have different abilities, but typically possess the required functionality to fulfill their design objectives.

Software agents should also have the ability to act autonomously without direct human interaction, be flexible, and in a multi-agent system, be able to communicate with other agents (being social). Agents are, to various degrees, aware of their environment, which also can be affected by the agents' actions. A mobile agent is a particular

class of agents with the ability during execution to migrate from one host to another where it can resume its execution. It has been suggested that mobile agent technology, amongst other things, can help to reduce network traffic and to overcome network latencies (Borselius,2002) .

In the future, agents will also be supplied with real money in some form to pay for resources or services (Sonntag & Hörmanseder, 2000).

When the Mobile Trade Agent migrates to unknown merchant server, there is a possibility of being attacked from that server (attacks on mobile agents by malicious hosts). Security is very important in mobile agent systems, both from the perspective of the agent as well as of the host. As mobile agents move to foreign hosts (which may not always be trusted or trustworthy), their data and code should be protected from tampering (Noordende, Overeinde, Timmer, Brazier & Tanbaum, 2007).

Confidentiality issues arise specially in the context of mobile agents carrying data that must be accessible only to specific, authorized hosts in their itinerary (Ametller, Robles & Ortega ,2004).

We focus on the security threats of a mobile agent. The mobile trade agent surf many sites which make him vulnerable to many attacks especially by malicious hosts. A malicious host is a system that tries to manipulate agent results or violate agent privacy and capture private information such as credit card number. So, the problem is how to make the mobile trade agent secure and avoid the attacks caused by a malicious host. We try to protect the agent by using cryptography algorithms to eliminate the basic attacks. In this thesis, we propose a

solution based on combination of public key authentication techniques and cryptography algorithms.

### **1.1.1 Agent Types**

Now, we address some definitions and activities related to Mobile Trade Agent and Agent Depository (AD).

The term **Agent** refers to one who acts on behalf of someone by his authority and trust in dealing with the business and others,(wikipedia).

#### **Software Agent:**

It is a piece of software that acts for a user or other program in a relationship of agency, which derives from the Latin agere (to do): an agreement to act on one's behalf. Such "action on behalf of" implies the authority to decide which (and if) action is appropriate. A **Software Agent comes on several types as follows,(wikipedia):**

- **Intelligent Agent:** an **Intelligent Agent is a** software agent that has some intelligence of learning and reasoning.
- **Autonomous Agent:** it is a type of agents which is capable of modifying the way in which they achieve their objectives.
- **Distributed Agent:** it is a type of agents which is capable of being executed on physically distinct computers.
- **Multi-Agent Systems:** these are distributed agents that do not have the capabilities to achieve an objective alone and thus must communicate.
- **Mobile Agents:** are agents that can relocate their execution onto different processors.

### **Agent Depository (AD):**

It is an agent holder, a depository, in which all agents are kept and interiorly sustained and controlled. The mobile trade agent is, by no means, stored at a user client computer, but the AD holds all Mobile Trade Agents. The user of the mobile trade agent is not passing its Mobile Agents straight to any server. It gives orders to the AD to simulate the mobile agent and order it to visit sites on the internet (Aqel, Aboud & Ahmed ,2007) .

#### **1.1.2 Available Mobile Agents Examples:**

In the following table, we summarize some of the Available Mobile Agents Examples:

<b>Product</b>	<b>Company</b>	<b>Language</b>
Agent Building Environment	IBM	C++, Java
Aglets	IBM Japan	Java
Concordia	Mitsubishi Electric	Java
Grasshopper	IKV++	Java
JADE	Tilab	Java
Microsoft Agent	Microsoft Corporation	Active X
Voyager	Object Space	Java

Table1: Available Mobile Agents Examples

### **1.1.3 Security Threats of Mobile Trade Agents :**

Mobile Trade Agent security problems can be divided into two categories:

(1) How can we protect agent against malicious host, and (2) how can we protect host against agent.

In our proposed system we will concentrate on protection of mobile agent against malicious host. Three fundamental problems of executing mobile code in an untrusted environment (Sander& Tschudin ,1998):

(i) Can a mobile agent protect itself against tampering by a malicious host? (Code and execution integrity).

(ii) Can a mobile agent conceal the program it wants to have executed? (Code privacy).

(iii) Can a mobile agent remotely sign a document without disclosing the user's private key? (computing with secrets in public).

A simple example often used to illustrate how a malicious host can benefit from attacking a mobile agent is the shopping agent as shown in Figure 1.1.

An agent is sent out to find the best airfare for a flight with a particular route. The agent is given various requirements, such as departure and destination, time restrictions, etc., and sent out to find the cheapest ticket before committing to a particular purchase. The agent will visit every airline and query their databases before committing to a purchase and reporting back to the agent owner (see Figure 1.1). A malicious host can interfere with the agent execution in several ways in order to make its offer appears most attractive.

For example, a malicious host could try to (Borselius,2002) :

- (1) Erase all information previously collected by the agent – in this way the host is guaranteed at least to have the best current offer.
- (2) Change the agent's route so that airlines with more favorable offers are not visited.
- (3) Terminate the agent to ensure that no competitor gets the business either.
- (4) Make the agent execute its commitment function, ensuring that the agent is committing to the offer given by the malicious host (if the agent is carrying electronic money, it could instead take it from the .In addition to this, the agent might be carrying information that needs to be kept secret from the airlines (e.g. maximum price).

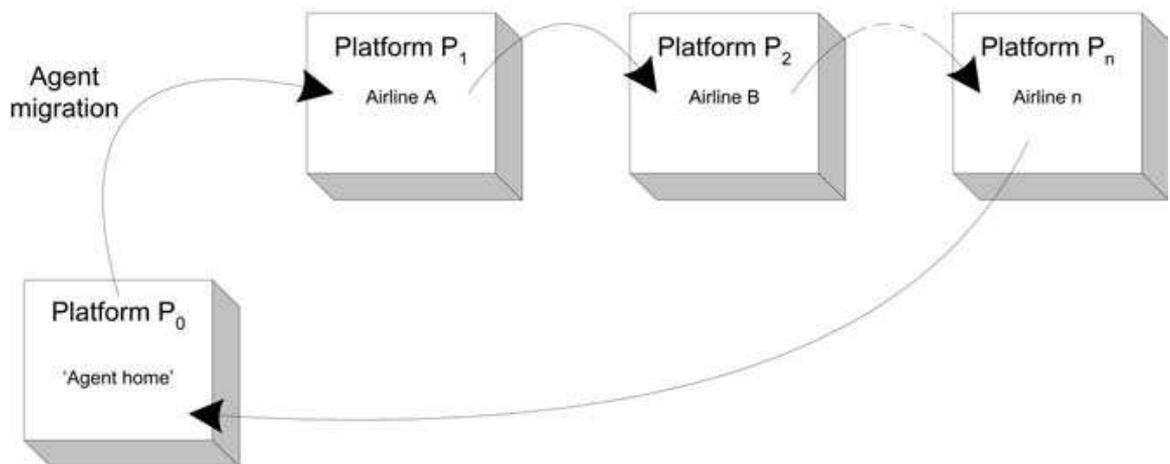


Figure 1.1 Shopping Agent, sent out to find best airfare.

## **1.2 Problem Definition**

Mobile agent technology has not become popular due to some problems such as security. The fact that computers have complete control over all the programs makes it very hard to protect mobile agents from untrusted hosts (Flocchini and Santoro ,2006).

Among the severe security threats faced in distributed mobile computing environments, two are particularly troublesome: harmful agent (that is the presence of malicious mobile processes), and harmful host (that is the presence at a network site of harmful stationary processes), (Lee, Alves & Harrison, 2004).

It is necessary to protect the mobile agent in terms of privacy and integrity against malicious-host.

There is no universal solution to the malicious host problem, but some partial solutions have been proposed. Many of the security mechanisms are aimed for detecting, rather than preventing, misbehaving hosts (Borselius,2002) .

Our security requirements can be attained via public key cryptography authentication techniques and cryptography algorithms, to make a secure mobile trade agent that can roam merchants' sites safely without the fear of host sever attacks.

### **1.3 Thesis Objectives**

- Our goal is to create a new technique in protecting mobile trade agent based on public key authentication techniques and cryptography algorithms such as RSA and Elgamal, which are known to be powerful algorithms.
- We also want to ensure that the mobile trade agent is secure enough to roam merchants' sites safely without the fear of host severers attacks.
- To make trade in mobile trade agent systems more efficient, secure, and attractive, allow the transactions to be accomplished in easy and safe way.
- This thesis provides a broader range of protection for mobile agents data, this work can serve as a contribution towards the security of e-commerce world.

#### **1.4 Thesis Motivations**

- A mobile trade agent has many advantages such as overcoming network latency and reducing the load on the network.
- In spite of mobile trade agent system benefits, we found it's not frequently used because of the security problem related to malicious hosts' attacks, so we thought that the mobile trade agent system will attract a large number of users if the system becomes more protected.
- To add new ideas and approaches in protecting mobile trade agent, using public key authentication techniques and cryptography tools to prevent the agent system's and the user's information from being revealed by the attackers.

#### **1.5 Thesis Contribution**

- This thesis proposed a mechanism that protects the information of mobile trade agent system from the malicious attacks and a two-phase encryption method, using public key encryption tool to ensure the information integrity. We believe that by encrypting the message into two phases; namely RSA encryption algorithm first and Elgamal second, the security level in mobile agent system will be increased.
- Truth to be said there is no guarantee that the system is secure enough from being attacked and keys never would be broken, but we see that our system is secure, since we used the most two powerful algorithms (RSA and Elgamal) it will cost the attacker long time to break our system protection mechanism and try to intercept and decipher the message sent by the agent.

- By increasing the security level in mobile agent system, a wide range of people would be attracted to use the mobile trade agent system. This would be a contribution to the security of e-commerce world.

### **1.7 Thesis Significance**

The significance of this thesis summarized by the following:

- Previous studies focused on protecting the host against hostile agent while our work tries to protect the mobile agent data against malicious host.
- Our work is analyzing the harmful host attacks on mobile agent and proposes solution to this problem based on public key authentication techniques and cryptography algorithms.
- User privacy and data integrity will be accomplished.

### **1.6 Thesis Organization**

This thesis consists of five chapters. The first chapter speaks of the thesis introduction and the objectives, also the reasons and motivation behind this work. The second chapter describes the related work and the previous studies of mobile agent system security.

The third chapter shows mobile-agent technology, discusses the architecture of mobile agent and how to prevent host attacks on mobile agent using RSA and Elgamal encryption algorithm .

The fourth chapter discusses the implementation of our proposed protection model mobile agents and the proposed scheme and how it works. Finally the last chapter summarizes the conclusion and the future work.

# *Chapter*

# *Two*

## **Chapter Two**

# **Literature Review and Related Work of Mobile Trade Agent Systems**

### **2.1 Introduction**

This chapter describes previous studies of mobile agent system security and provides an overview of related work and identifies the fundamental weaknesses in their approaches. The related works had been discussed first and related studies second.

### **2.2 Literature Survey Related to Mobile Agents Secure Migration**

Warnier, Oey, Timmer & Brazier (2007), introduced a mechanism to ensure that breach of integrity in migration paths of mobile agents in large scale distributed agent systems will be detected. This approach distributes trust over three hosts during each migration step. The combination of sequence numbers with signatures guarantees that one or more hosts can detect if part of the migration path, including cycles, has been removed.

Their approach assumes that a secure distributed mobile agent system provides the following basic properties: an agent runs on one single host at a time, is aware of its current host, and has the ability to migrate to other hosts in the system.

In addition the environment provides a public-key infrastructure for agents and hosts that they can be authenticated. The host, on which an agent is initialized, is assumed to be trusted by the agent's owner.

This host can be traced by all other hosts at any arbitrary moment in time. Agents preferably only migrate to trusted hosts. An agent's migration path provides means to detect breaches of integrity.

The simplest form of migration in a secure agent system requires sending and receiving hosts to mutually authenticate themselves using a PKI (**P**ublic **K**ey **I**nfrastructure), where PKI is method of using public and a private cryptographic key pair for message authentication or encrypting. The integrity of a migrating agent is ensured by having the sending host create a (digital) signature of an (hash of the) agent's code. This signature is transmitted together with an agent's code, and data (including state). The receiving host can then verify the integrity of an agent's code before re-initializing the agent process. If agents tend to disappear on one specific host, then this host is known to be unreliable, possibly malicious. A centralized trusted third party may prevent agents from migrating to untrusted and/or unreliable hosts (simply by not authorizing the migration).

The steps<sup>1</sup> below give a more detailed explanation of a migration step, using a TTP(**T**rusted **T**hird **P**arty), by an agent from host A to host B , see Figure 2.1:

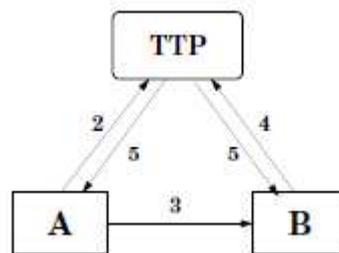
1. host A suspends and signs agent x: [x]A
2. host A reports to the trusted third party (TTP) that agent x will migrate from A to B. A sends [x]A along with the report.
3. host A sends agent x to host B
4. host B receives x and computes [x]B which it sends to the TTP.
5. The TTP verifies that A and B have both signed the same agent. If the verification passes, the TTP notifies both A and B that the

---

<sup>1</sup> A,B,C, denote hosts, small letters x,y,z, denote agents, arrows ( $\rightarrow$ ) represent migration steps between hosts and [x]A denotes the signature of agent x by host A.

migration has succeeded, and adds this migration step to the migration path it keeps for agent x.

6. host B starts the suspended agent.



**Figure 2.1. Agent migration using a central authority (trusted third party).**

The integrity of the migration path (item 1, above) is the basis for detecting malicious hosts and/or preventing them from doing any harm. Confidentiality (item 3) can be ensured by using encryption of sensitive data. Their main focus is the detection of breaches of integrity in migration paths of mobile agents and did not directly address the general problem of protection. Briefly, the algorithm works as follows: Suppose agent x migrates along the path  $A \rightarrow B \rightarrow C$ . Each migration step is recorded with the agent. Each step is signed by the host from which it's originated. When agent x migrates from B to C, host B asks host A to sign the migration step ( $B \rightarrow C$ ). The resulting signature is stored with the agent.

When host C receives the agent and the signatures, it confirms receipt to host A.

### **2.3 Protecting Mobile Agents using public key technique.**

Sameh and Fakhry (2002) present a three-tier approach which is a combination of code mess-up, encryption and limited lifetime of code and data (timing), that protect the agent code from malicious hosts'

attacks. The encryption algorithm used in the implementation is the DES algorithm. This algorithm is proved to have a reasonable key length, and is supported by the Java Security Classes. The goal of using encryption is to protect agent's important information which is stored in the list of prices that the agent collects from host to host. But, encryption alone does not guarantee a full protection for the agent.

If an agent expires, it can either be killed or recharged. Killing the agent will end its task completely. Sometimes an agent is delayed due to network problems, so killing an agent when its time expires will prevent it from performing its intended task. The problem with this approach is that the agent has to be assigned with a new expiration date and signed digitally by a party that the agent trusts.

Lee , Alves and Harrison (2004) proposed a security hybrid approach for mobile agents, which protect mobile agents from malicious hosts. The approach is mobile cryptography that encrypts mobile agents. Their approach implements mobile cryptography by proposing a hybrid method that merges a function composition technique (FnC) and some types of cryptosystem called homomorphic encryption scheme (HES), which allows direct computation on encrypted data. They produced a practical method of implementing mobile cryptography by extending Sander and Tschudin's idea<sup>2</sup> and developed a homomorphic encryption scheme. The biggest problem of Sander and Tschudin's approach was that there have been no published homomorphic encryption schemes to use in mobile cryptography. The

---

<sup>2</sup>Sander and Tschudin propose an approach based on the use of encrypted functions in which user encrypts a function  $s$ , then executed by the host, without the host having access to  $s$ . Although their approach is very promising, there is no secure implementation has been proposed as yet .

approach encrypts both code and data including state information in a way that enables direct computation on encrypted data without decryption. Their approach solves mobile agents' security problems like integrity and privacy. It prevents many types of privacy and integrity attacks, but, it cannot prevent blind modification attacks, which are a type of denial of service.

Aqel, Aboud and Ahmed (2007) introduced a scheme for mobile trade agent (MTA) that uses a combination of public key cryptosystem and distributed object technology. This distributed object technology makes MTA have possibility to roam the sites in the internet and protected within the defensive environments of the Agent Depository (AD). The Authorization Server (AS) is used for authorized transactions and pays the merchandise purchased by the MTA. They developed a scheme in which the MTA can supply a merchant the user smart card number. The merchant uses this data to demand payment from the Authorization Server (AS) which plays the bank role. The user smart card number is encrypted using a public key encryption scheme. The AS ensures that merchants only reclaim payment for merchandise purchased by the MTA and confirms that merchants receive the funds when the MTA purchases merchandise from them. Figure 2.2 illustrates the role of both AS and AD in the program. This MTA has more security prospects by using public key encryption schema to cipher the messages between entities and other agents on the internet. They introduce MTA that has the ability to act in behalf of the user, visiting internet sites, gathering related information of trade goods and where is best of them and their merchants. The MTA

has ability to make the e-commerce more open to market changes and increase profits.

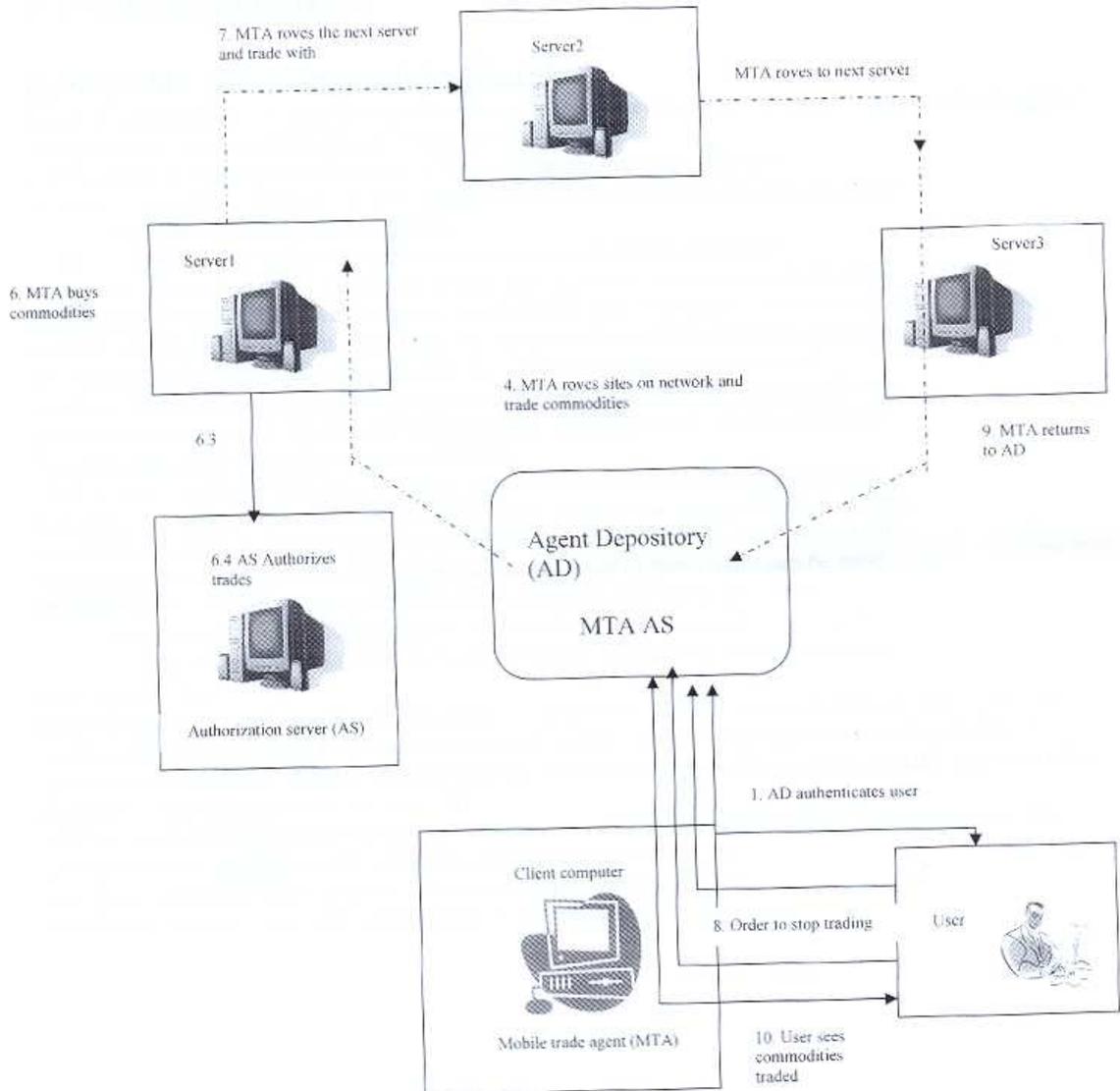


Figure 2.2: The MTA schema

## 2.4 Related Studies on Protecting Mobile Agents

Fischmeister (2000), conducted a study to discuss the mobile agent might steal resources, confidential data or use the server as starting point for a new attack. Accordingly many researchers devote their

time to this area. However, another main problem of mobile agent security is nearly neglected: the attack of a malicious server against a mobile agent. The server can steal resources and confidential data, too. Due to the fact that the server has in general no access restrictions, this problem is even harder to attack. The study presents a solution for this particular aspect of mobile agent security: the Supervisor-Worker Framework. The evaluation of the framework and the case study application showed that the framework effectively prevents tampering and eavesdropping and, additionally, boosts several other key aspects of mobile agents.

Singh (2000), conducted a study titled “Security of Mobile Agent”, suggests itself about the work emphasized in this thesis. The main area of discussion is the security of mobile agents on malicious host. Here a malicious host refers to a system in a network which can take advantage of the vulnerabilities of a mobile agent that has come to the host machine to get its work done. The study discusses about the mobile-agent, the various security threats that can be posed by malicious host and consequently the solutions. The study proposed a solution by combining few solutions and distilling the best from the solutions so that it can provide a better solution. Finally the study concludes with implementing the solutions and the results obtained by the experiments using trading example.

Chan (2000), conducted a study that discussed that Mobile software agents are emerging as a major trend of distributed systems in the near future. Different mobile agent frameworks are being actively developed in the research community. Looking forward, electronic commerce and information retrieval are two prospective directions for

application of mobile agents. Nevertheless, security and reliability are two crucial concerns for such systems, especially when they are to be used to deal with money transaction. In spite of some more classical reliability and security problems, attacks to agents by malicious hosts are a new and the most challenging part of the problem unsolved. The study showed that security and reliability issues of mobile agents, particularly in an electronic environment, are discussed. Models for mobile agent security and reliability have been developed, and a Shopping Information Agent System (SIAS) is built based as an experimental mobile agent application. Possible security attacks by malicious hosts to agents in the system are discussed, and specific solutions to prevent these attacks are devised. Security of the solutions is analyzed, and the performance overhead introduced is measured. Reliability problems of the system have been identified, and solutions implemented

Robles (2002) conducted a study to discuss agent technology, and showed that agents provide a further step in this direction and make possible new types of application, such as sea-of-data applications or specific pervasive computing. Nevertheless, the drawback of the new capabilities featuring this technology is the arising of new branches of security issues. It results hard to design security solutions for applications using mobile agents, especially in sea-of-data applications. There is not a definitive platform in which these applications are implemented and still offering security and ease to program. They present the start of the development of MARISM-A, **Architecture for Mobile Agents with Recursive Itinerary and Secure Migration**. This platform intends to observe commonly accepted agent standards FIPA(**F**oundation of **I**ntelligent **P**hysical **A**gents) and

MASIF (**M**obile **A**gent **S**ystems **I**nteroperability **F**acilities **A**gents), while providing flexibility to design secure sea-of-data applications.

The study trusts to find out the requirements of these new applications and presents a novel model of a methodology to achieve security solutions. The study applies the model to some scenarios of MARISM-A application. The same idea of using trust in sea-of-data applications can also be used to solve security issues in pervasive computing.

Koliousis (2005) conducted a study that discussed that the Mobile agents have several advantages over the traditional Client-Server, SNMP (*Simple Network Management Protocol*)-based approach in network management systems. However, the adoption of mobile agents in network management entails a number of security risks. They have developed a Java-based mobile agent infrastructure that enables the safe integration of mobile agents with the SNMP protocol. The Ajanta mobile agent environment has been used as the core component of our network management infrastructure, primarily because of its security model. The security of the system has been evaluated under agent to agent platform, and agent to agent attacks in order to prove the trustworthiness of mobile agents. A set of performance management scenarios have been simulated in order to show the correct workings of the system, but also to evaluate its performance.

## 2.5 The Public-Key Cryptography Concept

Cryptography is the most popular way to achieve the data security, in which the plain text is encrypted into cipher text before being transmitted. The cryptography systems can be divided into two types: symmetric key and asymmetric key cryptography.

The symmetric key cryptography uses one key for data encryption and decryption. Asymmetric key cryptography also called public key cryptography that uses two keys (see Figure 2.3), one of the keys, a public key is used for the plain text's encryption that is known to every one, the other key is a private key, which is used for cipher text's decryption, private key is hidden and kept secret from every one and only the receiver has it,(wikipedia).

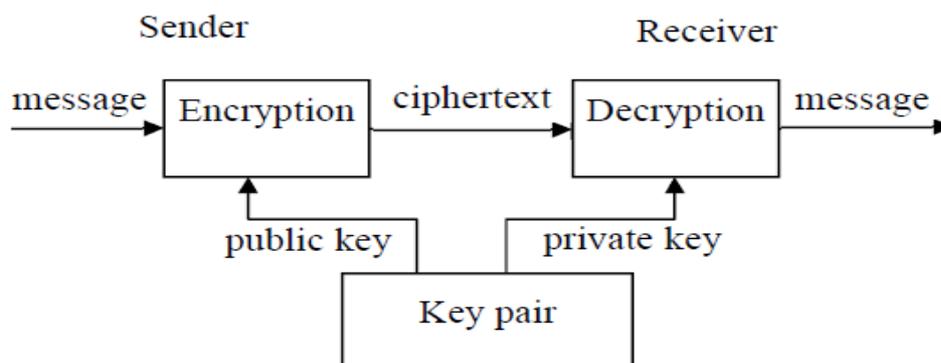


Figure 2.3 Public key cryptography

### 2.5.1 Advantages and Disadvantages of Public-Key Cryptography

- The main advantage of public-key cryptography is increasing security since the private keys never need to be revealed to anyone. In a secret-key system, by contrast, the secret keys must be transmitted (either manually or through a communication channel), and there may be a chance that an enemy can discover the secret keys during their transmission, (Arnaud, 1997).

- Another advantage of public-key systems is that they can provide a method for digital signatures. Authentication via secret-key systems requires the sharing of some secret and sometimes requires trust of a third party as well. As a result, a sender can repudiate a previously authenticated message by claiming that the shared secret was somehow compromised, by one of the parties sharing the secret. Public-key authentication, on the other hand, prevents this type of repudiation; each user has sole responsibility for protecting his or her private key. This property of public-key authentication is often called non-repudiation.

- A disadvantage of using public-key cryptography for encryption is speed: there are popular secret-key encryption methods that are significantly faster than any currently available public-key encryption method. Nevertheless, public-key cryptography can be used with secret-key cryptography to get the best of both worlds. For encryption, the best solution is to combine public- and secret-key systems in order to get both the security advantages of public-key systems and the speed advantages of secret-key systems. The public-key system can be used to encrypt a secret key which is used to encrypt the bulk of a file or message. Such a protocol is called a digital envelope.

- Public-key cryptography may be vulnerable to impersonation, however, even if users' private keys are not available. A successful attack on a certification authority will allow an adversary to impersonate whomever the adversary chooses to by using a public-key certificate from the compromised authority to bind a key of the adversary's choice to the name of another user,(Arnaud, 1997).

### 2.5.2 Introduction to RSA and Elgamal algorithms

Today, RSA and Elgamal are popular algorithms in public key cryptography system.

RSA stands for the initial of the last names of its inventors: **R**ivest, **S**hamir and **A**dleman, who first developed it in 1978. RSA security strength depends on the mathematical measures: the factoring problem and key size, choosing long key will increase the security by making it difficult to be revealed, RSA key size is between 1024 and 2048,(wikipedia).

<b>RSA Keys Length</b>	<b>The predicted Time</b>
1024 bit	2006-2010
2048 bit	2010-2030
3072 bit	beyond 2030

Table 2 : RSA key size and the predicated time to be breakable in it

Elgamal algorithm had been developed by Taher Elgamal in 1984, and named after him. Elgamal is based on the Diffie\_Hellman key exchange concept, which allows two parties (sender & receiver) to establish a shared secret key over an insecure communication channel.

Elgamal is probabilistic, in which a single plain text can be encrypted to many possible cipher texts, producing a cipher text that has the double size of the original plain text. Elgamal security strength depends on the difficulty of computing discrete logarithms.

*Chapter*

*Three*

## Chapter three

# Mobile Trade Agent Architecture and the Proposed Model

### 3.1 Introduction

This chapter shows and discusses the architecture of mobile agent and how to prevent attacks on mobile agent's information by using encryption tools.

### 3.2 Mobile Trade Agent Architecture

Mobile Trade Agent is one of the important features of e-commerce world. When a person wants to buy some product or service online, he or she may have no idea about the best product and where to buy it and there are many Internet sites to determine that. That is why the user sends his mobile agent to roam different sites for that purpose.

Mobile Trade Agent architecture can be thought as client-server model, the mobile agent can act as server send out by the user as a client to surf the Internet sites in order to get the requested information about a certain product.

The mobile trade agent can be viewed as a program that has the capability to make the right decisions, where to move on the Internet, collect and examine information about the visited merchants' sites.

Mobile Trade Agent system (as shown in Figure 3.1) works as follows(Aqel, Aboud & Ahmed ,2007) :

A. User sends order to the Mobile Trade Agent to roam internet sites and find information about product with best prices.

B. Mobile Trade Agent makes queries to Agent Depository about the possible sites to roam (ask about the server address that provides information about the product that the user asking for).

C. Agent Depository sends a list about the possible sites to roam to Mobile Trade Agent

D. Mobile Trade Agent roams server's sites and collects information and offers about the product.

E. Mobile agent gathers information and offers about the product from the visited sites.

F. After the Mobile Trade Agent evaluates the offers and product information, it sends evaluation report to the user.

The user receives the evaluated offer report and chooses the reasonable one. The user orders the mobile trade agent to purchase the product or goods.

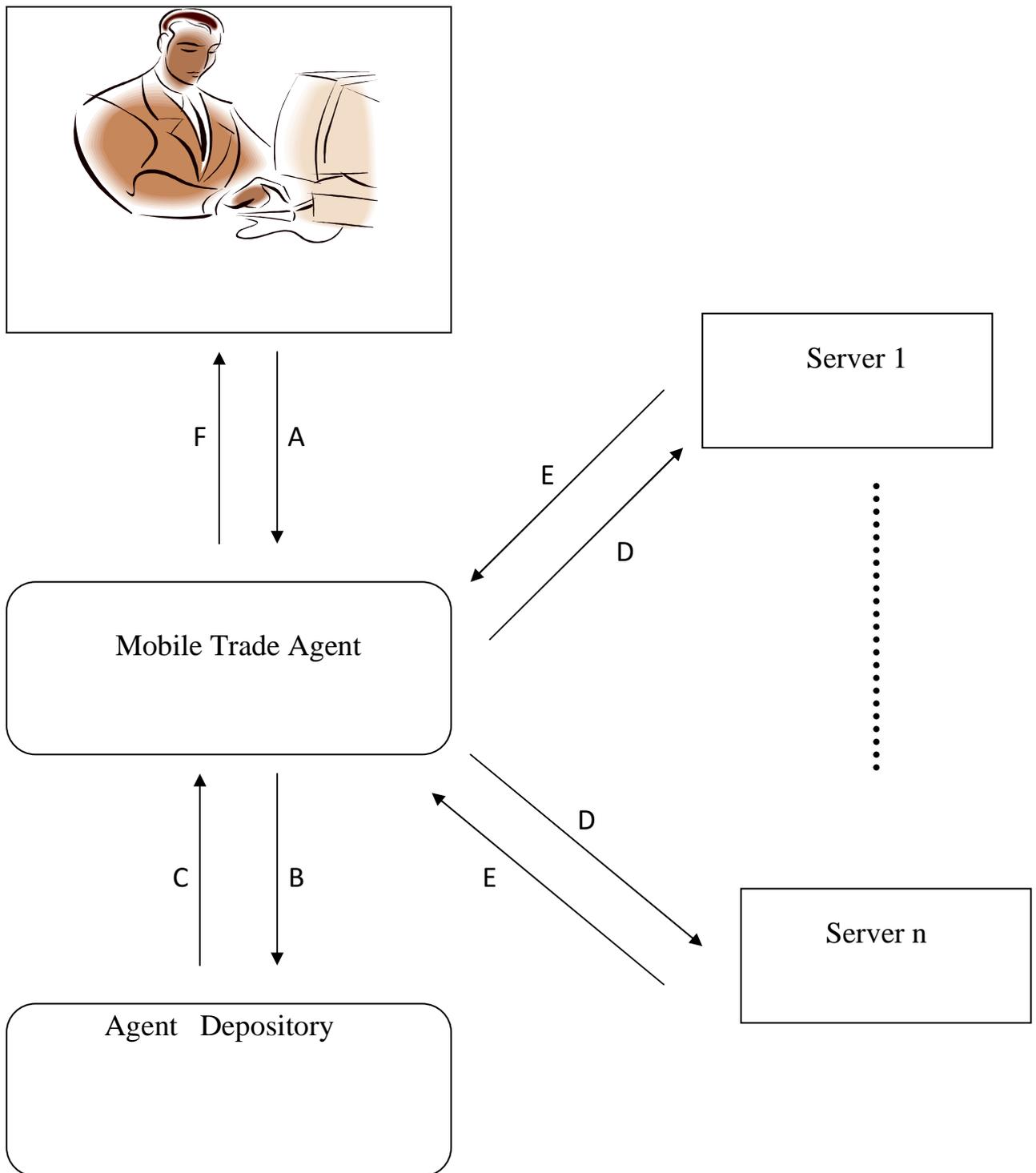


Figure 3.1: Mobile Trade Agent Schema

### **3.3 The Proposed Model**

When the mobile trade agent visits the merchant site and purchases the requested product on behalf of the user, the mobile agent supplies the merchant with the user credit card number.

Such important information like: credit card number, payment amount and products information without protection mechanism, could be vulnerable and in danger of being attacked by malicious host for example, who can manipulate and modify the agent's information about product and sites, tricks the gent by making him decide to buy such product (car, ticket ...etc) from the attacker's website.

We want for such sensitive information that is the mobile agent tries to hide it from the intruders, to be protected and make it venerable. In other words, we want to achieve data integrity and privacy. By integrity we mean the agent's information must not be manipulated and modified, on other hand, achieving the privacy means that the transmitted information must be kept from unauthorized person, which could be handled by using cryptography tools.

We add a protection mechanism in the mobile trade agent system (Figure 3.2), which is Encryption/Decryption, so when MTA roams sites and gets results and tries to send it to the client, this information encrypted using encryption algorithms before being send, the client decrypt the received information using decryption algorithms.

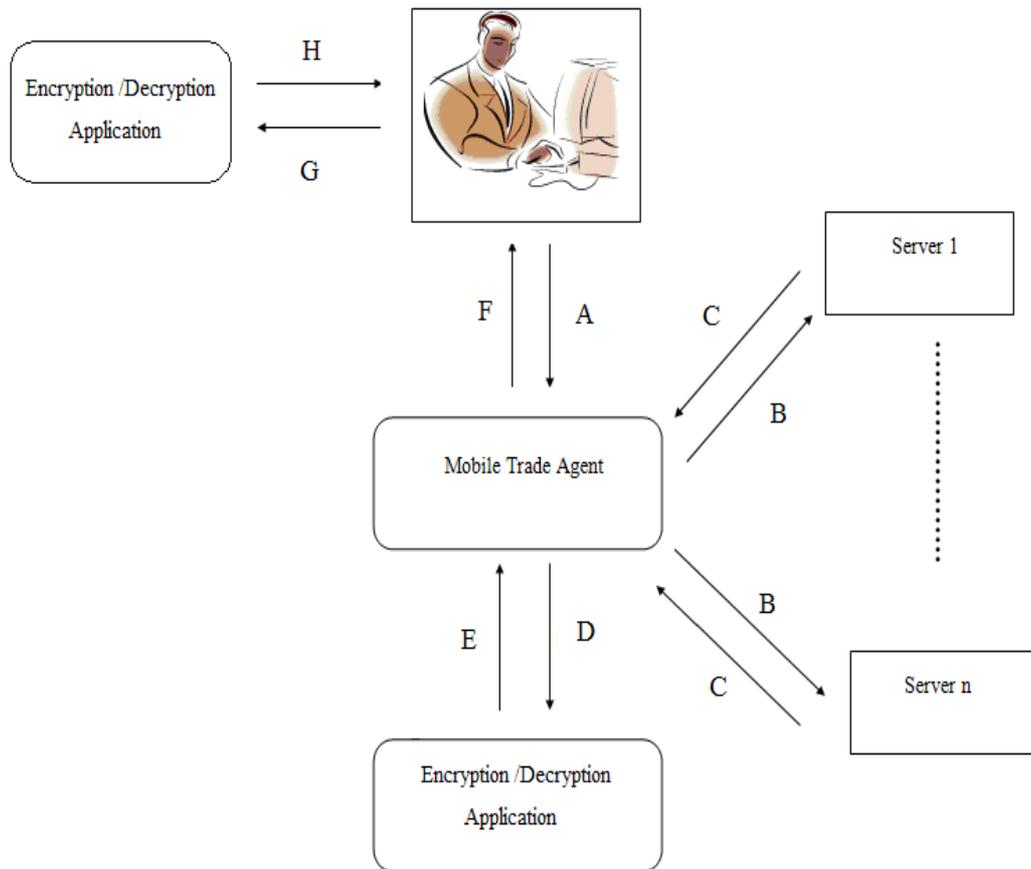


Figure 3.2 The Proposed Mobile Agent System Model

We will speak briefly about our proposed system which is much the same old mobile agent system in addition to the protection mechanism (Encryption/Decryption) that can be used for both sides: the sender (the mobile trade agent) and the receiver (the client). The system works as follows:

A. The client asks the Mobile Trade Agent to visit internet sites and find information about product with best prices.

B. Mobile Trade Agent roams server's sites and collects information and offers about the product.

C. Mobile agent gathers information and offers about the product from the visited sites.

D. After the Mobile Agent evaluate the offers and product information, encrypt this information using the system application (Encryption/Decryption).

E. The agent gets the encrypted information from the system application.

F. The agent sends the encrypted results to the client.

G. The client decrypts the received message using the system application (Encryption/Decryption).

H. The client gets the original message and read it and decides whether to buy the product or not, then the client sends his or her decision to the agent.

We believe that protection system will prevent attackers from knowing the agent's and client's information since it uses two algorithms, namely RSA and Elgamal in the encryption process.

Our proposed protection model has three major processes:

- 1) Key Generation : Public and Private Keys
- 2) Encryption: Two Phase Encryption, using RSA and Elgamal algorithms, in order encrypt the message to be sent to the client.
- 3) Decryption: Two Phase decryption, using Elgamal and RSA algorithms, in order to decrypt the message been received.

### **3.4 The Algorithms Used in The System**

In our system mobile agent's information is encrypted using asymmetric encryption algorithms; sometimes called Public Key encryption. It uses two keys, public key to encrypt the message, and private key to decrypt the message.

These algorithms used in the system are: RSA and Elgamal, these algorithms had been proven secure on the basis of the mathematical problem hardness that is difficult and time consuming to perform, such as : integer factoring and discrete logarithms. RSA is considered secure because of the hardness to find the factors of large prime numbers. For example the value of  $n = 923$ , it would take time to find the factors value for  $p = 71$  and  $q = 13$ . In fact, the larger value of  $n$  means the longer time to find the factors  $p$  and  $q$ .

Elgamal is considered secure because of the discrete logarithm, like the factoring problem, it's believed to be hard and difficult to compute.

#### **3.4.1. Generate keys**

RSA and Elgamal both have two key sets: public and private keys (Figure 3.3), so we need to generate two public keys of RSA & Elgamal and two private keys of RSA & Elgamal ,(wikipedia).

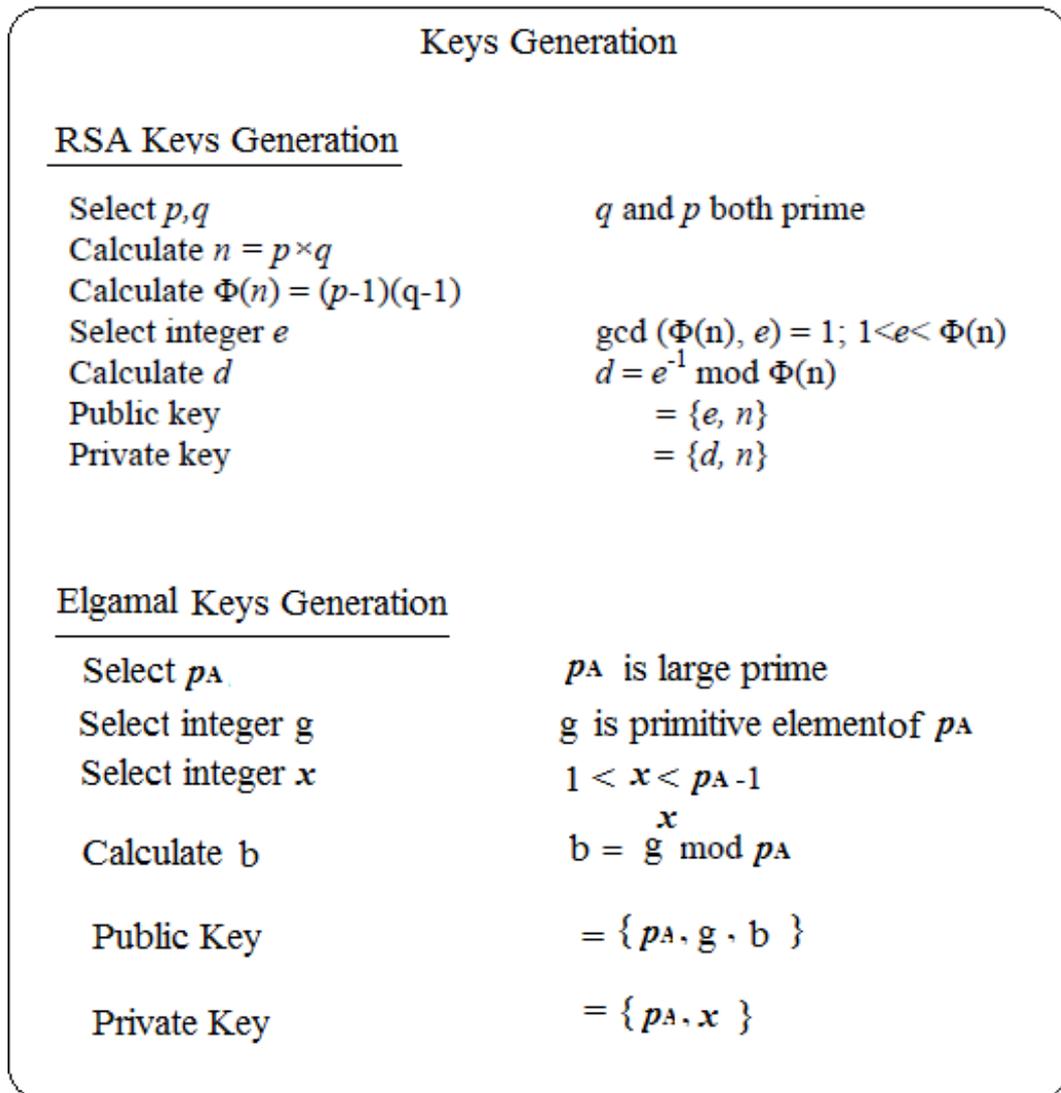


Figure 3.3: RSA & Elgamal keys generation scheme

#### 3.4.1.1. RSA Public and Private Keys Generation

RSA Public Key consists of two values:  $e$  and  $n$ , where  $n$  is a modulo, that is generated by multiplying large random prime numbers  $p$  &  $q$ , on the other hand,  $e$  is the public exponent,  $e$  and  $\Phi(n)$  are relatively prime, that is:

$$\gcd(e, \Phi(n)) = 1 \text{ and } 1 < e < \Phi(n).$$

$$n = p * q \quad (1)$$

$$\Phi(n) = (p-1) * (q-1) \quad (2)$$

$$ed \equiv 1 \pmod{\Phi(n)} \quad (3)$$

$$d = e^{-1} \pmod{\Phi(n)} \quad (4)$$

RSA Private Key consists of two values: d and n, where d is the private exponent, where d is kept secret and  $1 < d < \Phi(n)$ . For this example, the keys were generated as follows,(wikipedia):

1. Select two random prime numbers,  $p = 13$  and  $q = 7$ .
2. Calculate  $n = p * q = 13 \times 7 = 91$ .
3. Calculate  $\Phi(n) = (p-1) * (q-1) = 12 * 6 = 72$ .
4. Select e such that e is relatively prime to  $\Phi(n)$  and less than  $\Phi(n)$ ; in this case,  $e = 5$ .
5. Calculate  $ed \equiv 1 \pmod{\Phi(n)}$ , such that

$$d = e^{-1} \pmod{\Phi(n)} = 5^{-1} \pmod{72}$$

$$d = 29.$$

RSA Public Key (5, 91)

RSA Private Key (29,91)

### 3.4.1.2. Elgamal Public and Private Keys Generation

Elgamal Public Key consists of three values:  $pA$ ,  $g$  and  $b$ , where  $pA$  is a large prime random number,  $g$  is a primitive root of  $pA$  and  $b$  is calculated by  $g^x \text{ mod } pA$ .

$$b = g^x \text{ mod } pA \quad (5)$$

Elgamal Private Key consists of two values:  $x$  and  $pA$ , where  $x$  is the secret random number and  $1 < x < pA - 1$ . For this example, the keys were generated as follows,(wikipedia):

1. Select random prime number,  $pA = 23$ .
2. Choose a primitive root of  $pA$ ,  $g = 11$ .
3. Choose a secret random number  $x=6$ ,  $1 < x < 30$ .
4. Calculate  $b$ , such that  $b = g^x \text{ mod } pA$ .

$$b = 11^6 \text{ mod } 23 = 9$$

Elgamal Public Key (23,11,9)

Elgamal Private Key (23, 6)

### 3.4.2. Encryption Phase

In our approach the information or message to be sent to the client must be encrypted in two phases: use RSA first, and then use Elgamal second (Figure 3.4).

First we must represent the plaintext message as a positive integer and convert it using ASCII Code.

Before sending the message to the client, we must do the following:

1. Use RSA public key (e , n) to encrypt the original message.

$$m1 = m0 ^ e \text{ mod } n \quad \dots\dots ( 6)$$

2. Then use Elgamal public key ( $pA$ , g, b) to encrypt the result message after RSA encryption,  $m1$ .

The final form of the message after the encryption is  $m2$ , which has two values  $y1$  and  $y2$ . We can calculate those by the steps below:

- a) Select a random number  $r$  such that  $1 < r < pA - 1$

$$y1 = g^r \text{ mod } pA \quad ( 7)$$

- b) Find the value of  $y2$

$$y2 = (m1 * b ^ r) \text{ mod } pA \quad ( 8)$$

$$m2 = (y1, y2)$$

3. Send the  $m2$  to the client

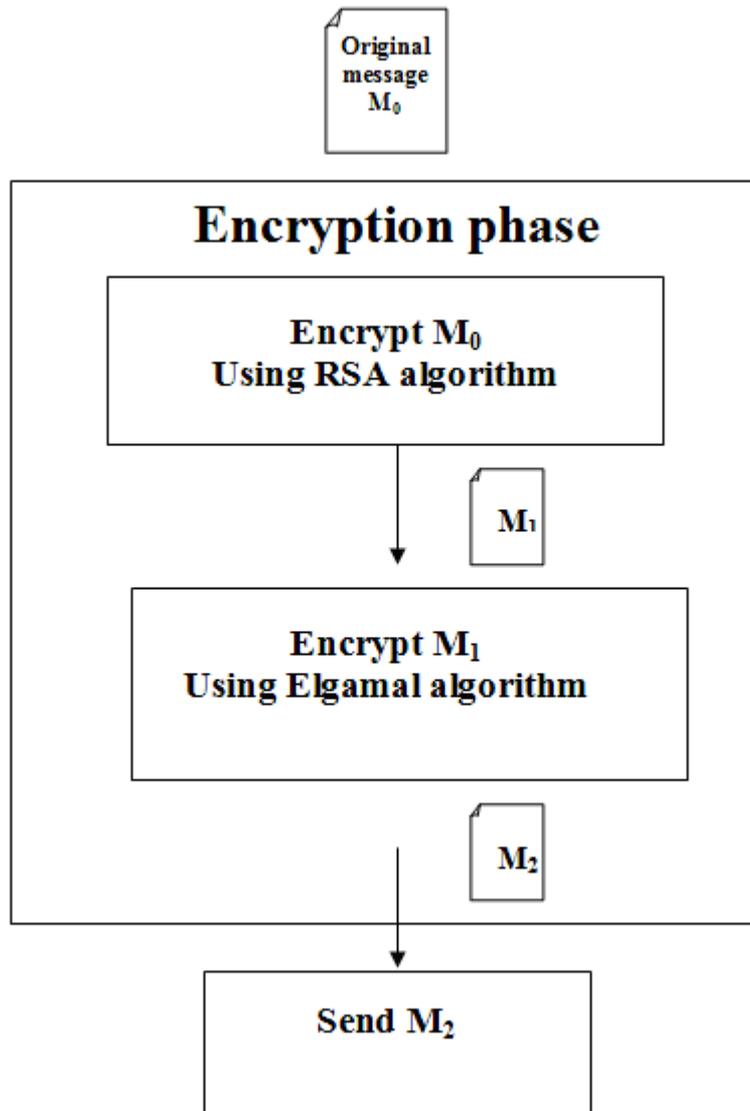


Figure 3.4 Two Phase Encryption

### 3.4.3. Decryption phase

The decryption mechanism is the reverse of the encryption mechanism. In order to decrypt the received message, the user or the client has two private keys (RSA & Elgamal) and he or she uses these for decryption process to have the original message as a result.

The client already has the encrypted message and attempt to decrypt it to retrieve the original message, This message for example may contain the agent information about best sites with the desired product.

The decryption phase works as follows (Figure 3.5):

1. Use Elgamal private key  $(pA, x)$  to decrypt the message  $m2(y1,y2)$ .

$$m1 = y2 * (y1)^{-x} \text{ mod } pA \quad (9)$$

2. Then use RSA private key  $(d, n)$  to decrypt the result message after Elgamal decryption,  $m1$ .

$$m0 = (m1^d) \text{ mod } n \quad (10)$$

The final form of the message after the decryption is  $m0$ , which previously had been converted into ASCII code character, after reversing this operation; the user has the original message. He or she can read it and decided what to do next, buy a certain good or refuse to buy it.

All of this shows the clear benefit of our approach, since we use two private keys, if the intruder intercept the message and attempt to

discover one of the private keys, how can he or she find the other private key. That will take along time and a lot of effort.

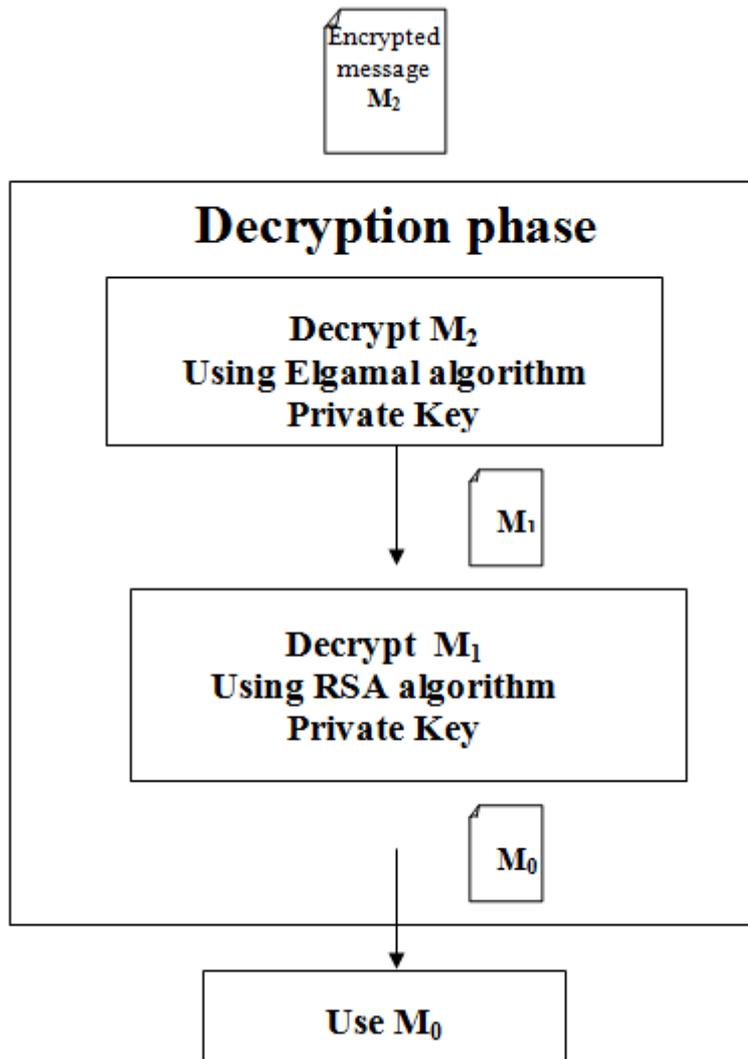


Figure 3.5 Two Phase Decryption

### 3.5 Worked Example

Here is an example that uses our encryption and decryption system. The parameters used here are small. To encrypt the message 'EADI' using the proposed security system, we must go through the following steps:

- **First generate Keys**

*Step1: Generate RSA algorithm Keys:*

1. Choose two different prime numbers, such as

$$p = 11 \text{ and } q = 7.$$

2. Compute  $n$

$$n = 11 * 7 = 77.$$

3. Compute  $\Phi(n)$

$$\Phi(n) = (11 - 1) (7 - 1) = 10 * 6 = 60$$

4. Choose  $e$  that:  $1 < e < n$  and  $\text{gcd}(e, \phi(n)) = 1$ .

$$e = 43.$$

5. Compute  $d$ , that  $d = e^{-1} \text{ mod } \Phi(n)$  and  $(d * e) \text{ mod } \Phi(n) = 1$

$$d = 7 \text{ since } e (43) * d (7) \text{ mod } \phi(n) (60) = 1.$$

The **RSA public key** is  $(e = 43, n = 77)$ .

The **RSA private key** is  $(d = 7, n = 77)$ .

***Step2: Generate Elgamal algorithm Keys:***

1. Choose prime number

$$p_A = 71$$

2. Select a primitive root of  $p_A$ ,  $g$ .

$$g=7.$$

3. Select a value  $x$  such that  $0 < x < p_A - 1$ .

$$\text{Let } x=11, \text{ where } 0 < 11 < 70.$$

4. Calculate  $b = g^x \bmod p_A$

$$b = 7^{11} \bmod 71 = 31.$$

The **Elgamal Public key** is ( $p_A=71, g=7, b=31$ )

The **Elgamal Private Key** is ( $p_A=71, x=11$ ).

- **Second The Encryption phases**

The system encrypts the message using the public keys of RSA and Elgamal.

Plaintext = EADI in ASCII code ('E' = 69, 'A' = 65, 'D' = 68, 'I' = 73)

The encryption has been done letter by letter:

***-Step 1: encrypt the message using RSA public keys***

***RSA encryption (Message) = (Message ^ e) mod n***

$$\text{RSA encryption (E)} = (69^{43}) \bmod 77 = 27$$

$$\text{RSA encryption (A)} = (65^{43}) \bmod 77 = 65$$

$$\text{RSA encryption (D)} = (68^{43}) \bmod 77 = 19$$

RSA encryption (I) =  $(73^{43}) \bmod 77 = 24$

**-Step 2: encrypt the RSA encryption results using Elgamal public keys**

**Elgamal encryption:**

$$(Message) y2 = (Message * b^r) \bmod pA$$

**Encrypted message = (y1, y2)**

-Select a random number r such that  $1 < r < pA - 1$ .

Let r=3, where  $1 < 3 < 70$ .

$$y1 = g^r \bmod pA$$

$$y1 = 7^3 \bmod 71 = 59$$

Elgamal encryption (27) =  $(27 * 31^3) \bmod 71 = 69$

Encrypted message (59, 69)

- r = 4.

$$y1 = 7^4 \bmod 71 = 58$$

Elgamal encryption (65) =  $(65 * 31^4) \bmod 71 = 69$

Encrypted message (58, 69)

- r = 5.

$$y1 = 7^5 \bmod 71 = 51$$

Elgamal encryption (19) =  $(19 * 31^5) \bmod 71 = 7$

Encrypted message (51, 7)

- r = 6.

$$y1 = 7^6 \bmod 71 = 2$$

Elgamal encryption  $(24) = (24 * 31^6) \bmod 71 = 20$

Encrypted message  $(58, 20)$

The final form of the message after encryption is  $\{(59, 69), (58, 69), (51, 7), (2, 20)\}$  and the system will send it to the user.

- **Third The Decryption phases**

*The user receives the encrypted message and decrypts it using his or her private keys.*

***Step 1: First decrypts the message using the Private key of Elgamal algorithm***

***Elgamal decryption (Message) =  $y_2 * (y_1)^{-x} \bmod pA$***

$(59, 69) \rightarrow$  Elgamal decryption (Message)

$$= 59 * (69)^{-11} \bmod 71$$

$$= 69 * 59^{(71-1-11)} \bmod 71 = 27$$

$(58, 69) \rightarrow 69 * 58^{(59)} \bmod 71 = 65$

$(51, 7) \rightarrow 7 * 51^{(59)} \bmod 71 = 19$

$(2, 20) \rightarrow 20 * 2^{(59)} \bmod 71 = 24$

***Step 2: decrypt the results using RSA Private Key***

RSA decryption (Message) =  $(\text{Message}^d) \bmod n$

Decryption  $(27) = (27^7) \bmod 77 = 69$

Decryption  $(65) = (65^7) \bmod 77 = 65$

$$\text{Decryption (19)} = (19^7) \bmod 77 = 68$$

$$\text{Decryption (24)} = (24^7) \bmod 77 = 73$$

The final form of the message after decryption is EADI.

### **3.6 System Application**

We have chosen C# programming language for our system implementation of the protection mechanism of our proposed model. C# is modern, high level, object oriented programming language. We saw that most current mobile trade agent systems had been implemented Using Java, but after we studied the features that C# is offering, we chose C# as our programming language.

### **3.7 System Requirements**

We have implemented the system using a modern computer that is capable of running Microsoft Visual Studios.Net 2008 and .NET Framework 3.5, that has some features which enable us write a simple and expressive code.

# *Chapter*

# *Four*

## Chapter Four

### Implementation and Evaluation of The Proposed Model

#### 4.2 Implementation of the Proposed Model

In this chapter the researcher discuss the implementation of our proposed protection system, how it works and evaluates it. And also we speak our system methods.

The system interface is one Form/Screen which represents the different processes and functions in the system. The system form is used for sides; sender and receiver (see the figure 4.1).

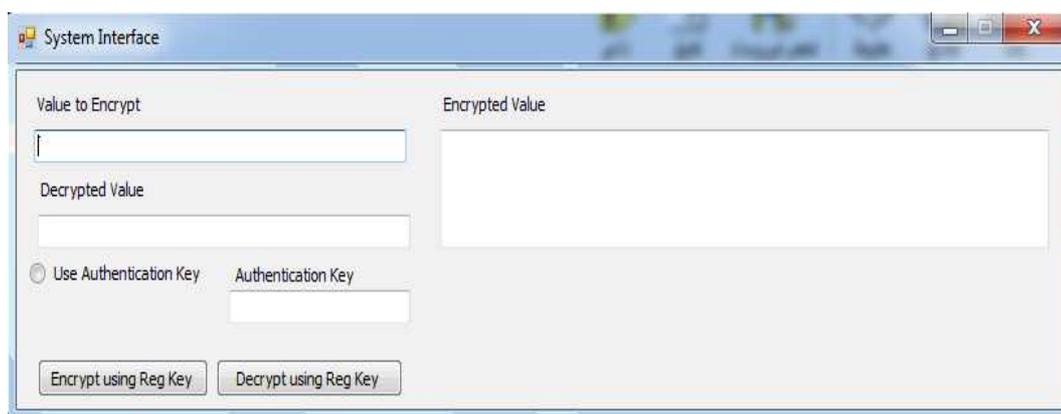


Figure 4.1 System Interface

We implemented our proposed system using C#, modern, scalable and simple programming language.

C# is a high level programming language based on C++, gathers programming eases of Visual Basic and the computing power of C++ and has the similarities of java features.

C# is simple but is a powerful language that enables the programmer to build a secure application.

C# is chosen as our programming language for our system implementation because of its portability features.

### 4.2.1 Authentication

Authentication is the most important issue of building a secure application; we need to be sure of the identity of the people with whom we will communicate and deal.

When you use the system interface, you identify yourself using authentication key (figure 4.2), a secret key that only you and the system know. By authentication key technique we will prevent unauthorized person from login to the application and use it.

After the user enters the authentication key, the system will check if the user is an authorized person to use the system .When the user has been authenticated, he or she will be allowed to use the system(figure 4.3).

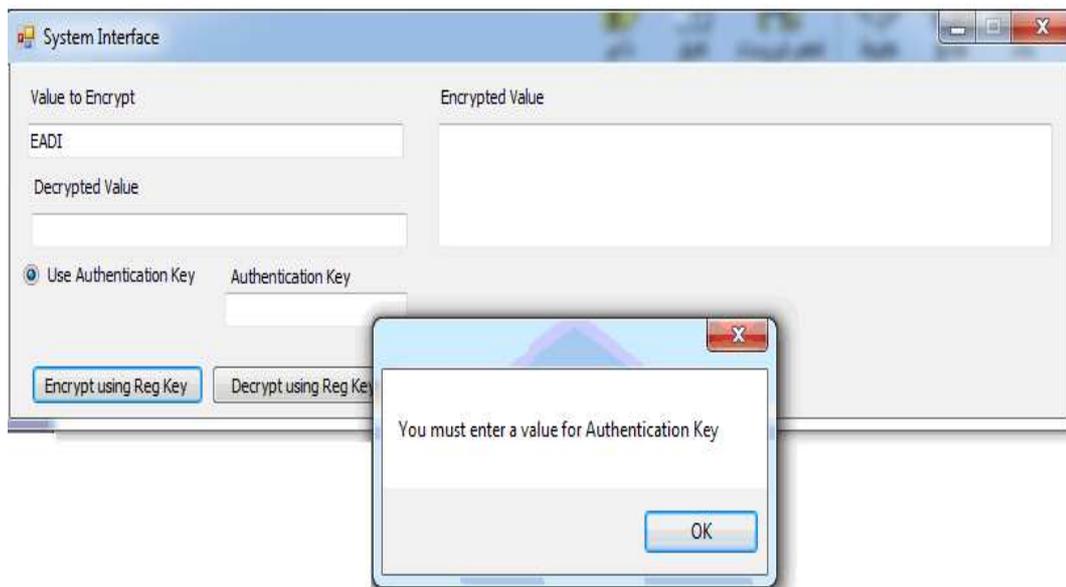


Figure 4.2: shows the user must enter the authentication key

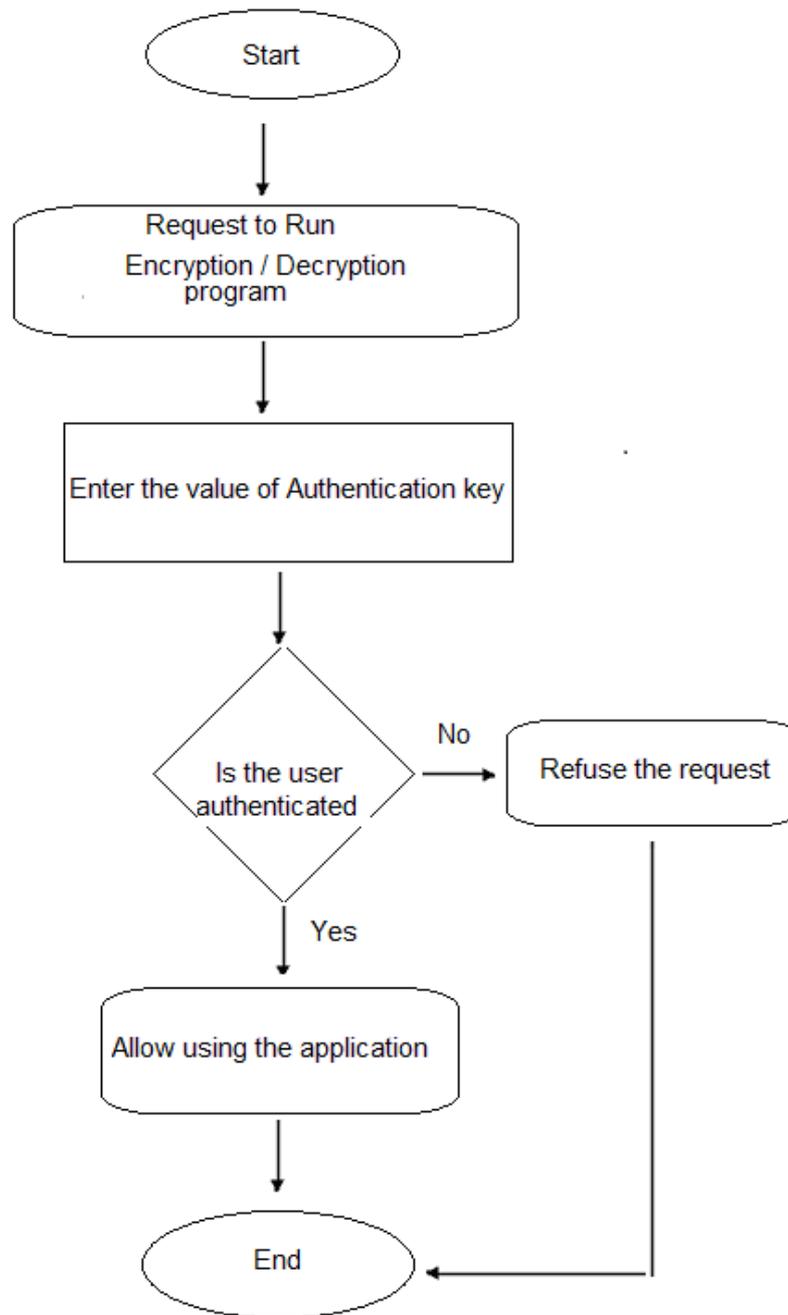


Figure 4.3 The System Flow Chart

## 4.2.2 Message Encryption

When the system user wants to encrypt the message before sending it to the receiver, he or she must do the following steps:

- The message should be printed in "Value to Encrypt" box.
- Click the button "Encrypt using Reg Key".
- The encrypted message will be appeared in "Encrypted Value" box.

See Figure 4.4 which shows that the message "EADI" is encrypted to:

"q5MXSiViaz1A01yvHfmlnf9FvQo18cRHOMNr01NEWRYsGxBx7ZJxZeuJgU4IQLX2xoaC3ZqOcyZlAKbVRyvJjchEO6iKSLImZdJ0J Ae++KU3omxC/rja/WfNjyMJFEKJ70+HsMIIdV+I/Gb2n3Y1NRbMK HRxoWanH4zdFAr4NfpY=".

After the message being encrypted, it will be sent to receiver.

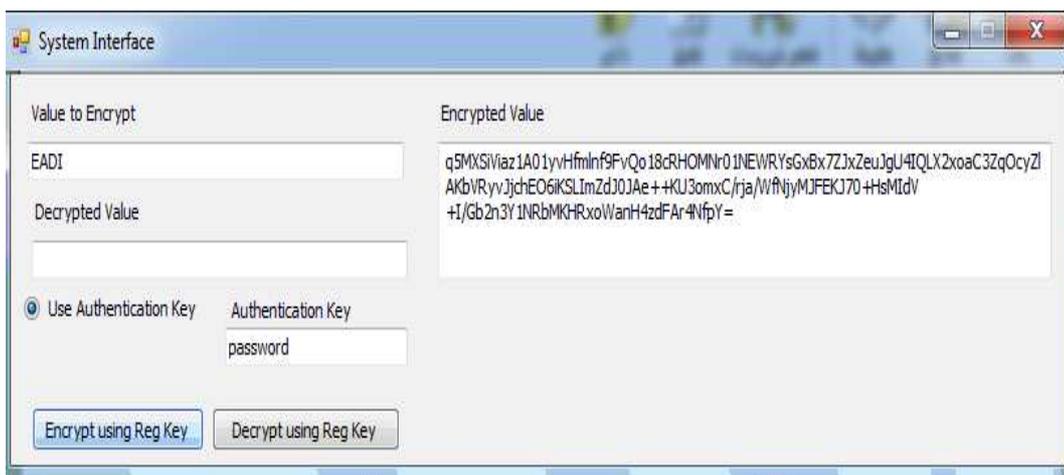


Figure 4.4 Message Encryption

### 4.2.3 Message Decryption

When the receiver has the encrypted message, he or she will decrypt the message by doing the following steps:

- Take the encrypted message and print it in "Encrypted Value" box.
- Click the button " Decrypt using Reg Key".
- The original message will be appeared in "Decrypted Value" box.

See Figure 4.5 which shows that the encrypted message (q5MXSiViaz1A01yvHfmlnf9FvQo18cRHOMNr01NEWRYsGxBx7ZJxZeuJgU4IQLX2xoaC3ZqOcyZlAKbVRyvJjchEO6iKSLImZdJ0JAe++KU3omxC/rja/WfNjyMJFEKJ70+HsMIdV+I/Gb2n3Y1NRbMKHRxoWanH4zdFAr4NfpY=) had been decrypted to the original message "EADI".

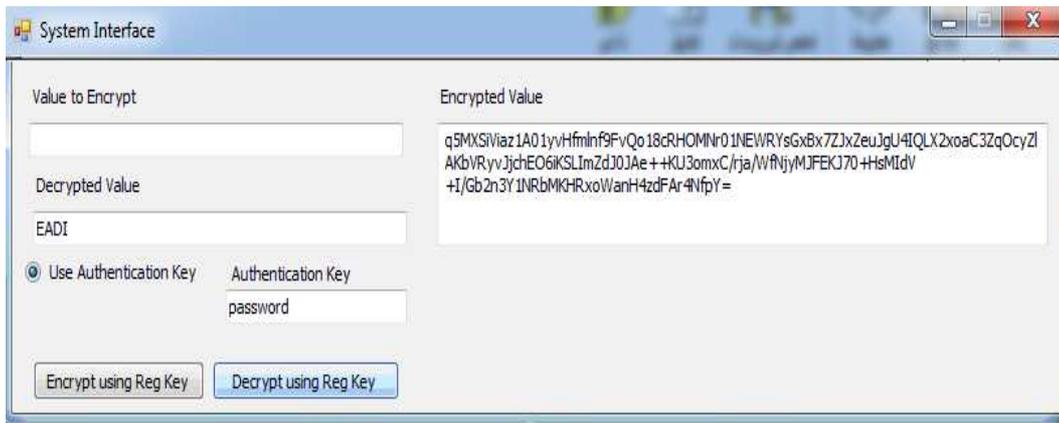


Figure 4.5 Message Decryption

### 4.2.4 The Application's Methods

Our system application has many programming methods. We will concentrate on the most important procedures such as:

- Generating the keys for RSA and Elgamal algorithms.
- Encryption processes.
- Decryption processes.

## 1. Generating the Keys

### a) First Generate RSA Key

The method below creates a key pair for you.

```

1 public AsymmetricCipherKeyPair GenerateKeys(int keySizeInBits)
2 {
3     RsaKeyPairGenerator r = new RsaKeyPairGenerator();
4     r.Init(new KeyGenerationParameters(new SecureRandom(),
5         keySizeInBits));
6     AsymmetricCipherKeyPair keys = r.GenerateKeyPair();
7     return keys;
8 }

```

### b) Second Generate ElGamal Key

```

private EG_Secret_Key GenerateKey(int nBits)
{
    BigInteger q = new BigInteger();
    BigInteger p;
    BigInteger g;
    BigInteger gPowTwo;
    BigInteger gPowQ;
    EG_Secret_Key eskKey = new EG_Secret_Key();

    q = BigInteger.genPseudoPrime(nBits - 1);
    p = BigInteger.genPseudoPrime(nBits);
    // find a generator
    do
    {
        g = new BigInteger();
        g = BigInteger.genRandom(nBits - 1);
        gPowTwo = g.modPow(new BigInteger(2), p);
        gPowQ = g.modPow(q, p);
    } while ((gPowTwo == 1) || (gPowQ == 1));

    BigInteger x;
    do
    {
        x = new BigInteger();
        x = BigInteger.genRandom(nBits);
    } while (x >= p - 1);

    BigInteger y = g.modPow(x, p);

    eskKey.p = p;
    eskKey.g = g;
    eskKey.x = x;
    eskKey.y = y;

    return eskKey;
}

```

### 3. Encryption Process

Now that we have a key pair, we are ready to encrypt using RSA then Encrypt Using Elgamal. In the example below, we use a public key to encrypt a byte sequence.

```
public byte[] Encrypt(byte[] data, AsymmetricKeyParameter key)
{
    RsaEngine e = new RsaEngine();
    e.Init(true, key);

    int blockSize = e.GetInputBlockSize();

    List<byte> output = new List<byte>();

    for (int chunkPosition = 0; chunkPosition < data.Length;
        chunkPosition += blockSize)
    {
        int chunkSize = Math.Min(blockSize, data.Length -
            (chunkPosition * blockSize));
        output.AddRange(e.ProcessBlock(data, chunkPosition,
            chunkSize));
    }
    return output.ToArray();
}
```

```
public override BigInteger[] Encrypt(BigInteger biInput, PublicKeyPacket pkpKe
{
    EG_Public_Key epkKey = new EG_Public_Key();
    epkKey.p = pkpKey.KeyMaterial[0];
    epkKey.g = pkpKey.KeyMaterial[1];
    epkKey.y = pkpKey.KeyMaterial[2];

    BigInteger k = new BigInteger();

    while (TextBox > (epkKey.p - 1))
    {
        k = new BigInteger();
        k = BigInteger.genRandom(epkKey.p.bitCount() - 1);
    }

    BigInteger B = epkKey.g.modPow(k, epkKey.p);
    BigInteger c = epkKey.y.modPow(k, epkKey.p);
    c = (biInput * c) % epkKey.p;
    //BigInteger c = (biInput * epkKey.y.modPow(k, epkKey.p)) % epkKey.p;

    BigInteger[] biOutput = new BigInteger[2];

    biOutput[0] = B;
    biOutput[1] = c;

    return biOutput;
}
```

#### 4. Decryption Process

Now from this procedure the application should decrypt the message using ElGamal Algorithm first using ElGamal's private key, then decrypting the results using RSA using RSA's private key.

```

}

public override BigInteger Decrypt(BigInteger[] biInput, SecretKeyPacket skpKey, string strPassphrase)
{
    BigInteger[] biKeyMaterial = skpKey.GetDecryptedKeyMaterial(strPassphrase);
    EG_Secret_Key eskKey = new EG_Secret_Key();
    eskKey.x = biKeyMaterial[0];
    eskKey.p = skpKey.PublicKey.KeyMaterial[0];
    eskKey.g = skpKey.PublicKey.KeyMaterial[1];
    eskKey.y = skpKey.PublicKey.KeyMaterial[2];

    if (biInput.Length != 2)
        throw new ArgumentException("biInput is not an ElGamal encrypted Packet");

    BigInteger B = biInput[0];
    BigInteger c = biInput[1];

    BigInteger z = B.modPow(eskKey.x, eskKey.p).modInverse(eskKey.p);

    BigInteger output = (z * c) % eskKey.p;

    return output;
}

```

```

public byte[] Decrypt(byte[] data, AsymmetricKeyParameter key)
{
    RsaEngine e = new RsaEngine();
    e.Init(false, key);

    int blockSize = e.GetInputBlockSize();

    List<byte> output = new List<byte>();

    for (int chunkPosition = 0; chunkPosition < data.Length;
        chunkPosition += blockSize)
    {
        int chunkSize = Math.Min(blockSize, data.Length -
            (chunkPosition * blockSize));
        output.AddRange(e.ProcessBlock(data, chunkPosition,
            chunkSize));
    }
    return output.ToArray();
}

```

### **4.3 Evaluation of the Proposed Model**

The system application behavior had been tested among sets of scenarios. There are some reasons that led us to believe in our system efficiency, how it provides and increases security level in mobile trade agent system.

- First, our system uses the authentication technique, in which unauthorized person will be prevented from using it.
- Second, the system uses the most two powerful cryptography algorithms: RSA and Elgamal. Both algorithms are based on complexity of its mathematical computations. As we know that the factorization (factoring large numbers) problem is the security strength basis of RSA algorithm as well as the Discrete Logarithm Problem is the security basis of Elgamal.
- Third, the system has two public keys (RSA and Elgamal) and two private keys, so if the attackers attempted to discover the private keys, he or she would face difficulty in revealing two important private keys, that he or she needs it decrypt the message which the mobile agent had sent it to the user. The attacker may be reveal one of the two private keys but there is a difficulty in finding two private keys.

In the context of this system, we focused on the security issues related to how to keep the information that the mobile agent tries to send it to the user from being manipulated or stolen by the system's attackers.

Since our protection system is based on using these two algorithms: RSA and Elgamal, we discussed some security issues related to these

algorithms, such as the key size that has a huge effect on message encryption and decryption process.

The execution time is also an important issue, since we use two algorithms together, this will affect on the execution time, the encryption process using RSA and Elgamal together increases the execution time and it's longer than using each one alone (see table 3), but we focus on the agent protection and security terms, so we tried to ignore that fact (see figure 4.6 & 4.7)

<b>Algorithm</b>	<b>Bit Length</b>	<b>Time in Millisecond</b>	<b>Time in Second</b>
RSA	4	5657	5.657
RSA	5	68057	68.057
RSA	6	509477	509.477
Elgamal	4	8491	8.491
Elgamal	5	95966	95.966
Elgamal	6	741218	741.218
RSA & Elgamal	4	10501670	10501.670
RSA & Elgamal	5	65401436	65401.436
RSA & Elgamal	6	99953320	99953.320

Table 3: The execution time of RSA & Elgamal algorithm

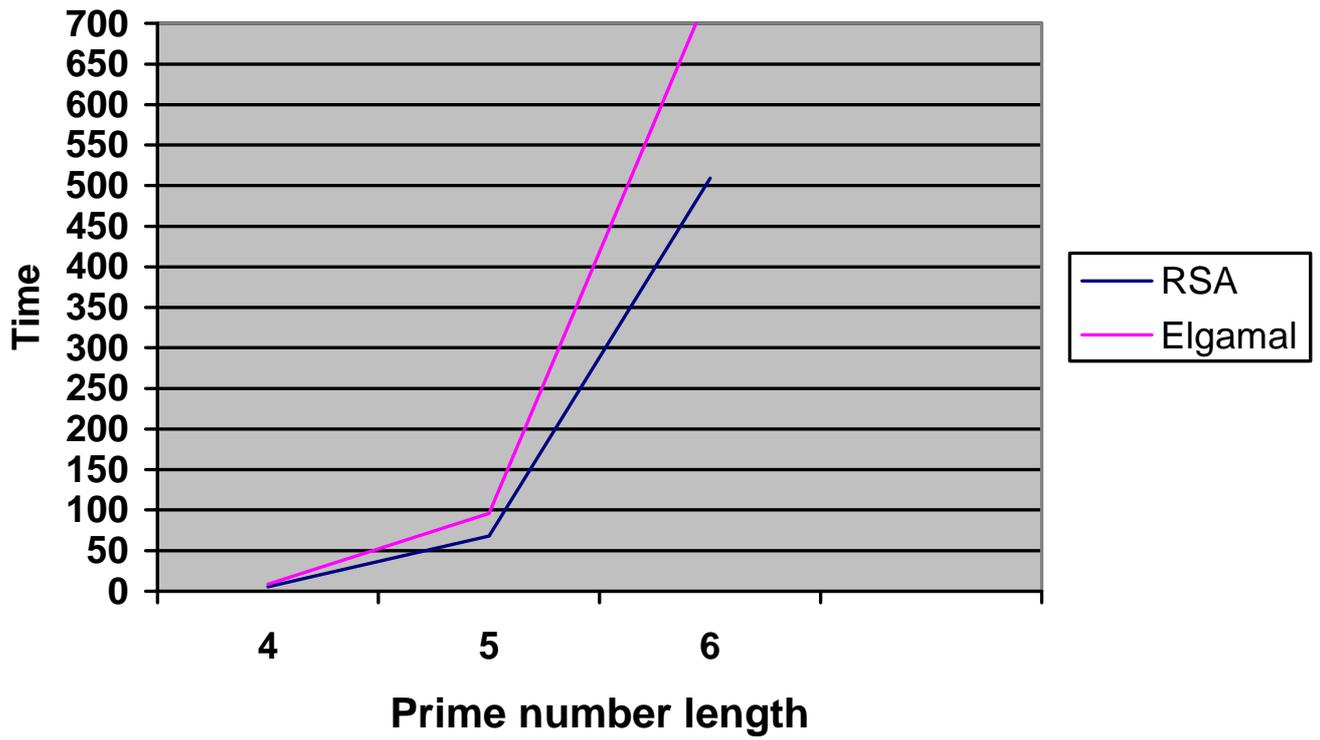


Figure 4.6 :The execution time chart of RSA & Elgamal

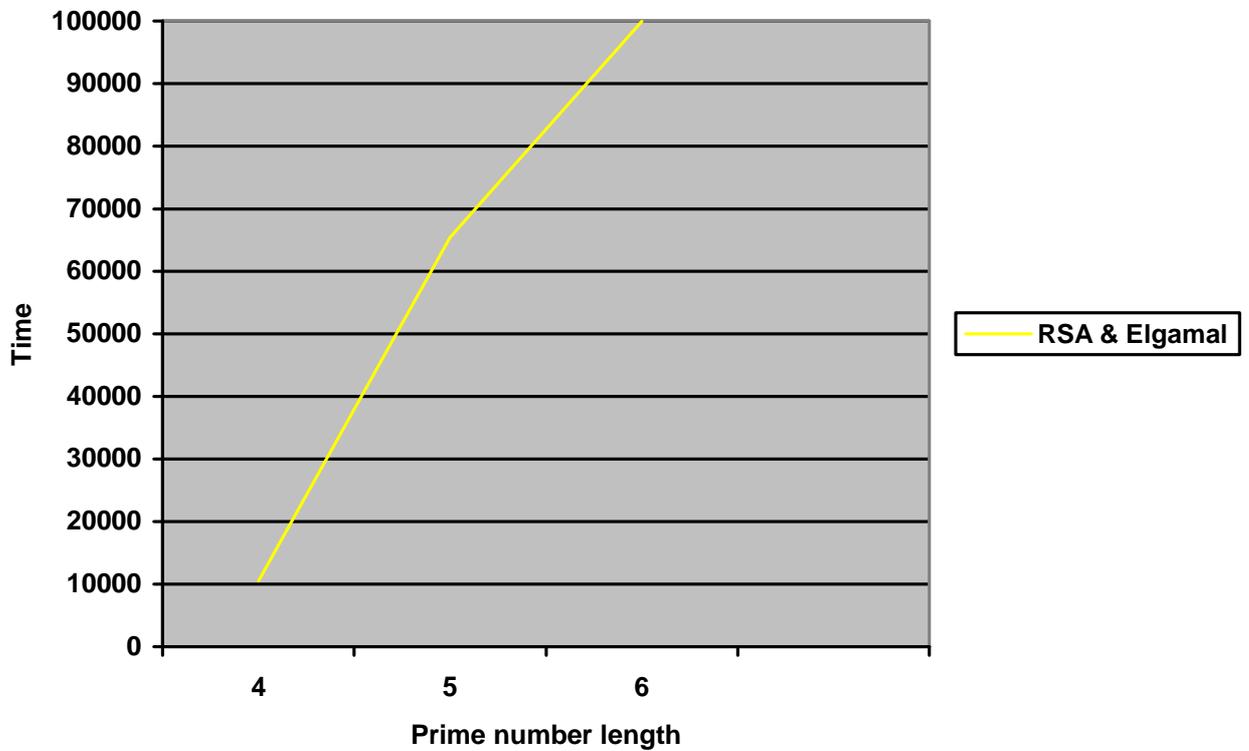


Figure 4.7: The execution time chart of RSA & Elgamal (together)

### 4.3.1 The Size of key

The importance of increasing the key length has important reflection on the security, which means increasing the difficulty for the message and private keys to be broken and decrypted.

In Elgamal, changing the size of prime number  $PA$  (*part of Elgamal public key*) will affect the size of the encrypted message ( $y_1, y_2$ ). So when the size (length) of  $PA$  has been increased, it means increasing the value of  $y_1$  and  $y_2$  of the encrypted message. As a result, the encrypted message size will be increased, and that makes it more difficult for the attacker to obtain original message.

In RSA, increasing the length of module  $n$  ( $n$  is generated by multiplying two random prime numbers  $p$  and  $q$ ) which is part of the public key means increasing the complexity of decomposing it into its factors ( $p$  and  $q$ ). This will result in increasing the values of the public, private key and the encrypted message. The larger value selection for  $n$  means larger size of encrypted message and makes it harder for the attacker to find the private key.

We also found that increasing the prime number length will affect in the execution time (see table 4 ), after using five different prime number in length , we saw that increasing in the prime length will increase the execution time (figure 4.7).

Prime number value	Bit Length	Time in Millisecond	Time in Second
1153	4	14487522	14487.522
14951	5	37661651	37661.651
101183	6	64241831	64241.831
1365071	7	72793339	72793.339
12657901	8	95967468	95967.468

Table 4: The execution time of prime numbers

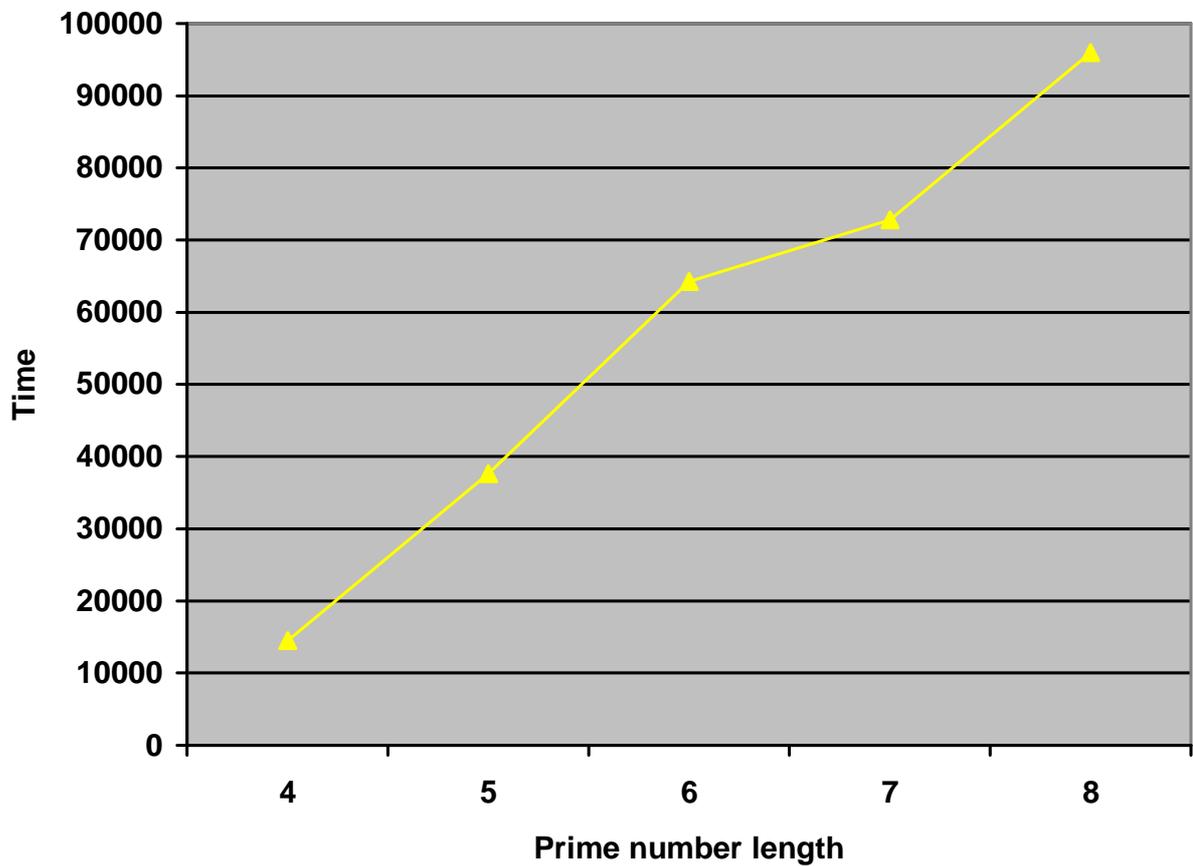


Figure 4.8 : The execution time chart of prime numbers

*Chapter*  
*Five*

## Chapter Five

### Conclusion and Future Work

#### 5.1 Conclusion

- Mobile agent systems can be more successful if the security problems had been solved. As a result of studying previous studies and related works, we saw that no one had come up with the perfect approach that can protect mobile agent, but on the other had no one had said that this is unsolved issue. As more attempts are being tried in this domain, we believed that the problem can be solved and simplified.
- This thesis had attempted to enhance the security and integrity aspects of mobile trade system. The agent's security problems had been discussed first, the mobile agent architecture had been studied and a protection mechanism had been designed and implemented.
- In the context of this thesis, we focused our attention on securing the agent's information, RSA had been chosen because of the factoring problem, and on the other hand Elgamal had been chosen because of the discrete logarithm problem.
- We think that our security system is a very promising approach, using a two-phase encryption/decryption mechanism.

- We tried to provide a better secure mobile agent system by using two cryptography algorithms RSA & Elgamal, which increase the security of mobile agent's transferred information, and make it harder to the attackers who try to steal and modify it. We showed that this is no longer easy to the attackers, because we join together the two properties of the two techniques, RSA and Elgamal, in order to make the attacking mission more difficult.

## **5.2 Future Work**

Our future works flow into two sections:

- In this thesis we used a two-phase encryption/decryption model, using RSA & Elgamal (see figure 3.4 & figure 3.5), and we suggest using the same used algorithms in our system but in the reverse order in both processes: encryption and decryption.
- Our future direction moves towards using a multiphase encryption/decryption model, using other algorithms, such as DES. We suggest studying the possibility of the multiphase encryption/decryption approach to discover the advantages of such approach and to see if that is possible to be applied or executed.

## References

- Amettler J., Robles S., Ortega J. A. , (2004). Self-protected mobile agents, *Third International Joint Conference on Autonomous Agents and Multiagent Systems* , Volume 1 (AAMAS'04).
- Aqel M.M., Aboud S.J. and AL-Fayoumi A. M., (2007). Secure mobile trade agent, *Journal of Computer Science*, 3(5): 329-334.
- Borselius N., (2002). Mobile agent security , *Electronics & Communication Engineering Journal* , Volume 14, no 5, pp 211-218.
- Chan H., (2000).*Mobile agent security and reliability issues in electronic commerce*,(A master dissertation), The Chinese University of Hong Kong.
- Fischmeister S.,(2000).*Building secure mobile agents*, ,(A master dissertation), Technical University of Vienna, Vienna.
- Flocchini P. and Santoro N. ,(2006). Distributed security algorithms by mobile agents , *In Proc. 8th International Conference in Distributed Computing and Networking*, (ICDCN 2006). LNCS 4308, pp. 1-14.
- Freeman A. & Jones A., (2003). *Programming .NET security*, (1<sup>st</sup> ed.) USA: O'Reilly.
- Koliousis A. K., (2005).*A trustworthy mobile agent infrastructure for network management*, (A master dissertation), University of Glasgow, Glasgow.
- Lee H., Alves J.and Harrison S. , (2004).The use of encrypted functions for mobile agent security , *In :Proceedings of the 37th Hawaii International Conference on System Sciences*, pp 10 .
- Mohamd A. S. and Fakhry D. ,(2002) .Security in mobile agent systems, In Proceedings of SAINT, *IEEE*, PP. 4-5.
- Noordende G.J., Overeinder B.J. , Timmer R. J., Brazier F. M., and Tanbaum A.S. ,(2007). A common base for building secure mobile agent middleware systems , *In: Proceedings of the International*

*Multi conference on Computer Science and Information Technology* ,Volume 2, pp. 13-25.

Robles s., (2002).*Mobile agent systems and a combined view toward secure sea of data application*, (A doctoral dissertation), The Autonomous University of Barcelona,Spain.

Sander T. and Tschudin C. F. ,(1998). Protecting mobile agents against malicious hosts, *International Computer Science Institute*, In Proceedings of Mobile Agents and Security, pp.44-60 .

Singel´ee D., Preneel B.,(2004).Secure e-commerce using mobile agents on untrusted hosts, *Computer Security and Industrial Cryptography (COSIC)*. article199 ,COSIC internal report 33 pages.

Singh Prabhat S., (2000). *Security of mobile agent*,(A master dissertation), Allahabad, Indian Institute of information Technology, Allahabad.

Sonntag M., Hörmanseder R. , (2000).*Mobile agent security based on payment* , *ACM Operating Systems Review*, v.34 n.4, p.48-55.

Warnier M., Oey M.A. , Timmer R.J. and Brazier F.M., (2007).Secure migration of mobile agents based on distributed trust, **In: Proceedings of the Tenth International Workshop on Trust in Agent Societies.**

Arnaud D.,(1997)."*What are the advantages and disadvantages of public-key cryptography over secret-key cryptography?*" , <http://ec.eurecom.fr/~arnaud/zds/appendix/node8.html>.

<http://en.wikipedia.org/wiki/Agent>,the page was last modified on 30 July 2011 at 19:24.

<http://en.wikipedia.org/wiki/Cryptography>, the page was last modified on 28 July 2011 at 20:18.

<http://en.wikipedia.org/wiki/RSA>, the page was last modified on 25 July 2011 at 15:08.

[http://en.wikipedia.org/wiki/ElGamal\\_encryption](http://en.wikipedia.org/wiki/ElGamal_encryption), the page was last modified on 22 May 2011 at 21:03.

## Appendix

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using ExtensionMethods;

namespace ExtensionSample
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            /// <summary>
            /// Responds to user clicking the radio buttons to
            enable/disable needed buttons
            /// </summary>
            private void RadioButtonHandler(object sender, EventArgs e)
            {
                btnEncryptUsingRegKey.Enabled = !rdoUseXml.Checked;
                btnDecryptUsingRegKey.Enabled = !rdoUseXml.Checked;

                btnEncryptUsingXml.Enabled = !rdoUseRegistry.Checked;
                btnDecryptUsingXml.Enabled = !rdoUseRegistry.Checked;
            }

            /// <summary>
            /// Responds to user clicking one of the encrypt buttons
            /// </summary>
            private void EncryptButtonHandler(object sender, EventArgs e)
            {
                // General validation
                if (string.IsNullOrEmpty(txtValueToEncrypt.Text))
                {
                    MessageBox.Show("You must enter a value to encrypt");
                    return;
                }

                // General validation
                if (string.IsNullOrEmpty(txtKeyInfo.Text))
                {
                    MessageBox.Show("You must enter a value for
Authentication Key");
                    return;
                }

                if (rdoUseXml.Checked)
                    // User wants to use a xml file value for the RSA key
                    txtEncryptedValue.Text =
txtValueToEncrypt.Text.EncryptStringUsingXMLFile(txtKeyInfo.Text);
                else
                    // User wants to use a registry value for the RSA key

```

```

        txtEncryptedValue.Text =
txtValueToEncrypt.Text.EncryptStringUsingRegistryKey(txtKeyInfo.Te
xt);
    }

    private void DecryptButtonHandler(object sender, EventArgs e)
    {
        // General validation
        if (string.IsNullOrEmpty(txtEncryptedValue.Text))
        {
            MessageBox.Show("You must have encrypted a value before
you can decrypt");
            return;
        }

        // General validation
        if (string.IsNullOrEmpty(txtKeyInfo.Text))
        {
            MessageBox.Show("You must enter a value for Authentication
Key");
            return;
        }

        try
        {
            // Make sure user has not messed with the encrypted value
            string
            byte[] testConversion =
Convert.FromBase64String(txtEncryptedValue.Text);
        }
        catch
        {
            MessageBox.Show("The encrypted value does not appear to be
in a valid format");
            return;
        }

        if (rdoUseRegistry.Checked)
            txtDecryptedValue.Text =
txtEncryptedValue.Text.DecryptStringUsingRegistryKey(txtKeyInfo.Te
xt);
        else
            txtDecryptedValue.Text =
txtEncryptedValue.Text.DecryptStringUsingXMLFile(txtKeyInfo.Text);
    }

    private void Form1_Load(object sender, EventArgs e)
    {
    }
}
}

using System;
using System.IO;
using System.Security.Cryptography;

```

```

namespace ExtensionMethods
{

    public static class StringExtensionMethods
    {

        /// <summary>
        /// </summary>
        /// <param name="encryptValue"><see cref="System.String"/>
value to encrypt</param>
        /// <param name="publicKey"><see cref="System.String"/>
registry key name that contains the public key</param>
        /// <returns></returns>
        public static string EncryptStringUsingRegistryKey(this string
encryptValue, string publicKey)
        {
            string encryptedValue = string.Empty;

            if (string.IsNullOrEmpty(publicKey))
                throw new ArgumentNullException("You must provide the name
of the registry key for the public key");

            CspParameters csp = new CspParameters(1);

            csp.KeyContainerName = publicKey;

            // Supply the provider name
            csp.ProviderName = "Microsoft Strong Cryptographic
Provider";

            try
            {
                //Create new RSA object passing our key info
                RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider(csp);

                // Before encrypting the value we must convert it over to
byte array
                byte[] bytesToEncrypt =
System.Text.Encoding.UTF8.GetBytes(encryptValue);

                byte[] bytesEncrypted = rsa.Encrypt(bytesToEncrypt,
false);

                // Extract our encrypted byte array into a string value to
return to our user
                encryptedValue = Convert.ToBase64String(bytesEncrypted);
            }
            catch (CryptographicException cex)
            {
                Console.WriteLine(cex.Message);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }

            return encryptedValue;
        }
    }
}

```

```

    /// <summary>
    /// Encrypts the specified string value using RSA encryption
    /// </summary>
    /// <param name="encryptValue"><see cref="System.String"/>
</param>
    /// <param name="publicKeyPath"><see cref="System.String"/>
key</param>
    /// <returns></returns>
    public static string EncryptStringUsingXMLFile(this string
encryptValue, string publicKeyPath)
    {
        string encryptedValue = string.Empty;

        string pubKey;
        if (string.IsNullOrEmpty(publicKeyPath))
            throw new ArgumentNullException("You must provide the path
to a xml file for the public key");

        // Read public key from xml file
        using (StreamReader reader = new
StreamReader(publicKeyPath))
        {
            pubKey = reader.ReadToEnd();
        }
        CspParameters csp = new CspParameters(1);

        try
        {
            //Create new RSA object passing our key info
            RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider(csp);

            // Load our public key data
            rsa.FromXmlString(pubKey);

            // Before encrypting the value we must convert it over to
byte array
            byte[] bytesToEncrypt =
System.Text.Encoding.UTF8.GetBytes(encryptValue);

            byte[] bytesEncrypted = rsa.Encrypt(bytesToEncrypt,
false);
            encryptedValue = Convert.ToBase64String(bytesEncrypted);
        }
        catch (CryptographicException cex)
        {
            Console.WriteLine(cex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }

        return encryptedValue;
    }

    /// <summary>
    /// </summary>

```

```

    /// <param name="decryptValue"><see cref="System.String"/>
value to decrypt</param>
    /// <param name="publicKey"><see cref="System.String"/>
key</param>
    /// <returns></returns>
    public static string DecryptStringUsingRegistryKey(this string
decryptValue, string privateKey)
    {
        // This is the variable that will be returned to the user
        string decryptedValue = string.Empty;

        // Make sure user supplied a value for the registry key
        if (string.IsNullOrEmpty(privateKey))
            throw new ArgumentException("You must provide the name
of the registry key for the public key");

        CspParameters csp = new CspParameters(1);

        // Registry key name containing the RSA private/public key
        csp.KeyContainerName = privateKey;

        // Supply the provider name
        csp.ProviderName = "Microsoft Strong Cryptographic
Provider";

        try
        {
            //Create new RSA object passing our key info
            RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider(csp);

            byte[] valueToDecrypt =
Convert.FromBase64String(decryptValue);

            byte[] plainTextValue = rsa.Decrypt(valueToDecrypt,
false);

            decryptedValue =
System.Text.Encoding.UTF8.GetString(plainTextValue);

        }
        catch (CryptographicException cex)
        {
            Console.WriteLine(cex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }

        return decryptedValue;
    }

    /// <returns></returns>
    public static string DecryptStringUsingXMLFile(this string
decryptValue, string privateKeyPath)
    {
        // This is the variable that will be returned to the user
        string decryptedValue = string.Empty;

```

```

// Variable to hold contents of private key xml
string privateKey;

// Make sure user supplied a value for the registry key
if (string.IsNullOrEmpty(privateKeyPath))
    throw new ArgumentNullException("You must provide the name
of the registry key for the public key");

// Read public key from xml file
using (StreamReader reader = new
StreamReader(privateKeyPath))
{
    privateKey = reader.ReadToEnd();
}

CspParameters csp = new CspParameters(1);

// Supply the provider name
csp.ProviderName = "Microsoft Strong Cryptographic
Provider";

try
{
    //Create new RSA object passing our key info
    RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider(csp);
    rsa.FromXmlString(privateKey);

    byte[] valueToDecrypt =
Convert.FromBase64String(decryptValue);
    byte[] plainTextValue = rsa.Decrypt(valueToDecrypt,
false);

    decryptedValue =
System.Text.Encoding.UTF8.GetString(plainTextValue);
}
catch (CryptographicException cex)
{
    Console.WriteLine(cex.Message);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

return decryptedValue;
}
}

```

```

    }
}

public override BigInteger Decrypt(BigInteger[] biInput, SecretKeyPacket skpKey, string strPassphrase)
{
    BigInteger[] biKeyMaterial = skpKey.GetDecryptedKeyMaterial(strPassphrase);
    EG_Secret_Key eskKey = new EG_Secret_Key();
    eskKey.x = biKeyMaterial[0];
    eskKey.p = skpKey.PublicKey.KeyMaterial[0];
    eskKey.g = skpKey.PublicKey.KeyMaterial[1];
    eskKey.y = skpKey.PublicKey.KeyMaterial[2];

    if (biInput.Length != 2)
        throw new ArgumentException("biInput is not an ElGamal encrypted Packet");

    BigInteger B = biInput[0];
    BigInteger c = biInput[1];

    BigInteger z = B.modPow(eskKey.x, eskKey.p).modInverse(eskKey.p);

    BigInteger output = (z * c) % eskKey.p;

    return output;
}

public override BigInteger[] Encrypt(BigInteger biInput, PublicKeyPacket pkpKey)
{
    EG_Public_Key epkKey = new EG_Public_Key();
    epkKey.p = pkpKey.KeyMaterial[0];
    epkKey.g = pkpKey.KeyMaterial[1];
    epkKey.y = pkpKey.KeyMaterial[2];

    BigInteger k = new BigInteger();

    while (k.BitCount > (epkKey.p - 1))
    {
        k = new BigInteger();
        k = BigInteger.GenerateRandom(epkKey.p.BitCount() - 1);
    }

    BigInteger B = epkKey.g.modPow(k, epkKey.p);
    BigInteger c = epkKey.y.modPow(k, epkKey.p);
    c = (biInput * c) % epkKey.p;
    //BigInteger c = (biInput * epkKey.y.modPow(k, epkKey.p)) % epkKey.p;

    BigInteger[] biOutput = new BigInteger[2];

    biOutput[0] = B;
    biOutput[1] = c;

    return biOutput;
}
}

```