Journal of King Saud University – Computer and Information Sciences (2014) 26, 131–142

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

ORIGINAL ARTICLE

# Fragmentation based encryption approach for self protected mobile agent

**Shashank Srivastava** *, **G.C. Nandi**

*Indian Institute of Information Technology, Allahabad, India*

Received 31 December 2012; revised 7 April 2013; accepted 22 August 2013
Available online 31 August 2013

**KEYWORDS**

AES;
Fragmentation based encryption;
Mobile code;
Mobile agent;
Self protected mobile agent;
Formal modelling

**Abstract** Distributed applications provide challenging environment in today's advancing technological world. To enhance the aspects of better performance and efficiency in real scenario mobile agent's concept has been brought forward. As every technological movement is aligned with its repercussions, the mobile agent technology also has its inherent security loopholes. Usage of agent technology poses various security threats over networked infrastructure. Moreover numerous researches have already been proposed to take the edge off inherent security risk faced by mobile agent, but all these approaches did not resolve the malicious execution environment problem in permissible and effectual conduct.

Gaining the understanding of mobile agent architecture and the security concerns, in this paper, we proposed a security protocol which addresses security with mitigated computational cost. The protocol is a combination of self decryption, co-operation and obfuscation technique. To circumvent the risk of malicious code execution in attacking environment, we have proposed fragmentation based encryption technique. Our encryption technique suits the general mobile agent size and provides hard and thorny obfuscation increasing attacker's challenge on the same plane providing better performance with respect to computational cost as compared to existing AES encryption.



## 1. Introduction

Mobile agent technology entices various distributed applications due to its positive features like intelligence, autonomy, adaptability, flexibility, etc. (Lange and Oshima, 1999), but security issues and its overheads are shading down its global acceptance. In mobile agent paradigm, various research approaches have been highlighted in past decade analyzing the security concern but still malicious execution environment is a bottleneck for its deployment on wide scale. The major threat for the implementation of this technology is the modification or analysis of the agent's code and critical data sent over the platform at the execution instance. Mobile agent freely roams across network from one execution environment to another and executed there and the execution platform which executes the mobile agent could try to perform malicious activity (Jansen and Karygiannis, 2000; Borselius, 2002).

* Corresponding author. Tel.: +91 9984905199.
E-mail address: shashank12march@gmail.com (S. Srivastava).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

1319-1578 © 2013 Production and hosting by Elsevier B.V. on behalf of King Saud University.
http://dx.doi.org/10.1016/j.jksuci.2013.08.002

In agent based infrastructure, two entities participate in whole communication and computation. One is home platform where the agent owner is responsible for agent creation; other is host platform which executes the agent. Malicious entity can perform various attacks on agent's component when agent traverses in communication channel. So security over communicational channel needs to be established. Secondly when an agent migrates from one execution environment to other, it provides its control to the execution environment which makes it vulnerable to different types of security attacks that could be performed through the execution environment (Borselius, 2002; Sander and Tschudin, 1998; Macrides, 2002).

Our research provides a new direction to the agent based applications primarily targeting confidentiality concerns of agent security during communication channel or on execution platform vulnerable to the following security threats:

- Insecure communication channel
- Malicious execution environment

Generally encryption is provided to maintain the confidentiality factor of Information. Encryption using single shared key known as symmetric encryption is used for sending the large amount of data. It is computationally efficient but fails to provide authentication and integrity. To overcome the said challenges and enhance security, asymmetric encryption concept was built in but it lagged in providing efficient computational factor.

Analyzing the encryption techniques in the environment of mobile agent where the agent code moves towards data location, our research work gives a new direction to traditional security approach. As in agent paradigm, the code size is very less so the issues like network overhead, bandwidth and latency are no more challenges. Henceforth our main intent to enhance security in terms of time and space for the agent and the agent based infrastructure. We have proposed a protocol where agent is structured in modules for performing various activities. Self decryption module and fragmentation based encryption are the back bone of our security protocol.

Our research is organized in following way. Section 2 analyses the previous researches conducted in related area of agent security and tries to find out the current state of art in the security prospect. Section 3 proposes a security solution in the form of security protocol and presents a light weighted fragmentation based encryption technique. Section 4 provides a formal verification of security threats and our security protocol. Section 5 deals with implementation details of protocol with results. At the last, in Section 6, we summarize our security protocol and algorithm and focus on its future scope.

## 2. Analysis of previous researches

Security is the biggest concern which darkens the advantageous side of mobile agent infrastructure. As soon as the mobile agent migrates from home platform, it goes out of the control of its owner that gives the opportunity to attacker (eavesdropper, network sniffer, execution environment, etc.). In the case, if the execution environment is malicious, it can perform various attacks on mobile agent. Mainly the following are the threats for such traversing of mobile agent in an untrusted environment.

- Analysis of code to change its execution behavior.
- Modification of code.
- Analysis of data collected during execution of agent's itinerary.
- Modification or deletion of data collected.

So in a nutshell, there are basically two types of attacks which can be performed by execution environment.

- Execution privacy
- Execution integrity

We listed out following directions analyzed for coming to the point of current security requirements that need to be sorted out for the further enhancement of agent based paradigm in terms of security and performance.

In mobile agent system, code, data and execution state migrate from one execution environment to another. During execution of mobile agent, execution platform has full control over agent's code, data and execution state. Hohl (Hohl, 1998) proposed obfuscation technique in which mobile agent code is scrambled in such a manner that no one can understand it easily, like in java, java complier converts .java file to .class file which is written in bytecode. Java bytecode runs by JVM which is platform independent. In the case of mobile code, java bytecode moves from one host to another host (class serialization). Java byte code is not in readable form. But now there are various decompilaters are available to convert java bytecode to java program. In the same way, java deobfuscator is also available which helps in deobfuscating the obfuscated program (Armoogum and Caully, 2011).

Moreover, java obfuscation only provides execution privacy or confidentiality to the mobile code but it cannot be ensured that the particular obfuscated code is deobfuscatable or not. In this case, if execution platform has sufficient computational capacity, it can deobfuscate the code before executing the code. This technique is a preventive measure and its security is dependent upon the computational capacity of execution environment.

The second prominent solution is encrypted key function. Encrypted function computation approach was first forwarded by Sander and Tschudin (Sander and Tschudin, 1998) in 1997 which states code is hidden during execution to achieve privacy. However this mobile cryptography technique was strong enough to provide execution privacy but it only worked for the special case of polynomial and rational function. Besides this, no homomorphism encryption system exists till now which could help to implement mobile cryptography in a practical environment.

In this solution (Lee, 2004), home platform has an algorithm which computes a function $f$. At remote site, the target host has an input $x$ and it computes $f(x)$ to provide services to agent. In order to secure the function $f$ so that remote host cannot read this, home platform encrypts the function $f$ to get $E(f)$ and then embodies encrypted function within program. Home platform inserts this program within agent's code and sends it to remote host platform for execution. The target platform runs program on input $x$ and produces $E(f(x))$ then the result is sent back to its home platform. Home platform decrypts it and gets $f(x)$. This mechanism enables the agent to execute in a secure manner at remote untrusted platforms.

In the year 2004, Lee (Lee, 2004) brought a novel hybrid concept of functional composition and homomorphism encryption scheme which worked like the original mobile cryptosystem in a way of encrypted data computation with decryption. In the year 2004, Ametller and Robles (Ametller et al., 2004) gave a breakthrough idea for securing mobile agent to build self protected mobile agent. In their scheme, agents carry their fully protected encapsulated protection mechanism to protect their code and data. After this innovative breakthrough in the area of agent security, various researchers used it as foundation and gave their idea for agent based application. This technique promotes agent driven approach, i.e. security verification is performed by agent itself and it also uses a cryptographic interface between agent and execution platform so that platform could not directly interact with the agent. The need for agent to gain access to the private key of the execution platform is one point which makes this approach sometime inferior.

The fourth area of security solution is based on the co-operation of mobile agents where functionality is distributed among different mobile agent so that the attacker could not compromise the information by compromising a single mobile agent. First Roth (Roth, 1998, 1999) gave the concept of co-operating agents to build the secure mobile agent system. After that Samuel Pierre (Benachenhou and Pierre, 2006) proposed a solution that is based on the perfect co-operation of a sedentary agent running inside a trusted third host. However this technique is good enough to provide security by establishing communication between agent platform and trusted server but each platform has to give the detail of agent execution to the trusted server through a large number of communication link established which creates extra burden (Ouardani et al., 2007; El Rhazi, 2007).

The fifth area of clone agent protocol is somehow related to sedentary agent approach, but here the clone agent is executed on the same execution host before critical code execution. Thus, instead of exposing the mobile agent to the malicious host, it first sends a clone to investigate the behavior of the execution environment. In CAP (Bouchemal et al., 2009; Benachenhou and Pierre, 2006), once an attack is detected, the mobile agent puts the responsible host in its malicious host list and changes its destination accordingly.

Tarig Mohamed Ahmed has introduced a new mechanism called Secure-Image Mechanism (SIM) Ahmed, 2009 to protect the mobile agent from malicious hosts. In this technique, secure image controller creates the image of mobile agent and sends this to unknown host for execution. Host executes this image and sends it back to the secure image controller where verification of agent's integrity takes place by comparing it with original copy of the agent. If it finds any malicious modification, SIM presumes the host to be malicious. This mechanism provides security but takes much time and creates unnecessary communication overhead in network.

Sixth direction is related to cryptographic protocols (Guan and Zhang, 2010; Srivastava and Nandi, 2011a,b) that is the very basic and prominent solution for providing security to any area of communication. All other approaches integrate various cryptographic algorithms and protocols to accomplish their security needs. In general, in order to encrypt large size data, symmetric encryptions are used and to distribute symmetric key, asymmetric encryptions like RSA are preferably used.

After doing exhaustive analysis of security solutions and current status of malicious host problem, we bring forth a solution which is motivated from the four prominent approaches of self protection, obfuscation cryptographic algorithms and co-operations of different modules of agent.

## 3. Proposed security protocol

According to self protection technique, there are two types of security for malicious host problem. One is platform driven security in which platform is responsible for verifying the security operations like signature verification, agent decryption, etc. On the second aspect, agent driven security in which, agent itself is responsible for all security operations, i.e. agent decrypts itself and then executes itself. Our proposed protocol focuses on obfuscation and self protection concept (Ametller et al., 2004).

Our protocol first uses the concept of obfuscation but in a novel manner. First agent owner creates mobile agent and breaks in into different byte arrays such that first byte is placed in first array, second byte of code in second array, third byte of code in third array and so on. Other types of distribution can also be performed. Agent owner generates a key on the basis of this code can be reassembled to its execution state. Actually the code is distributed into different byte arrays according this key only. In this way, the platform which wants to execute this agent first has to find the reassembling key $K$ and then according to this, assembles the code and then it will be able to execute it. So agent owner needs to secure reassembly key $K$.

The main objective of obfuscation is that no one can understand the code as a whole because before execution, the code is scattered into different byte arrays according to specified order ($K_R$). Agent owner provides privacy to the code in communication channel from intruder or man in middle attack without encrypting whole code.

In order to secure reassembly key $K_R$, agent owner encrypts it with symmetric key $K$ which is the function of $f(K_1)$. And for the secure distribution of this symmetric key $K$, Agent owner encrypts $K_1$ with the public key of next host so that only the intended recipient can decrypt it. After scattering mobile agent, agent owner creates controller agent which embeds scattered mobile agents and all security measures (like reassembly algorithm, reassembly key, etc.).

Main controller agent has following components:

1. Scattered mobile agent in the form of different byte arrays and reassembly algorithm.
2. Encrypted Reassembling key.

Controller code communicates with the execution platform to get key component $K_1$. After getting key $K_1$, controller code calculates decryption key $K$, i.e. $f(K_1)$. This decryption key $K$ is responsible for the decryption of encrypted reassembly key. (Refer Fig. 1.)

### 3.1. Proposed protocol

1. Agent owner creates mobile agent $\boldsymbol{M}$ and scatters it into different byte arrays according to key specified by agent owner ($K_S$). $K_S$ is scattering key which is used to scatter the whole mobile agent into different byte arrays.

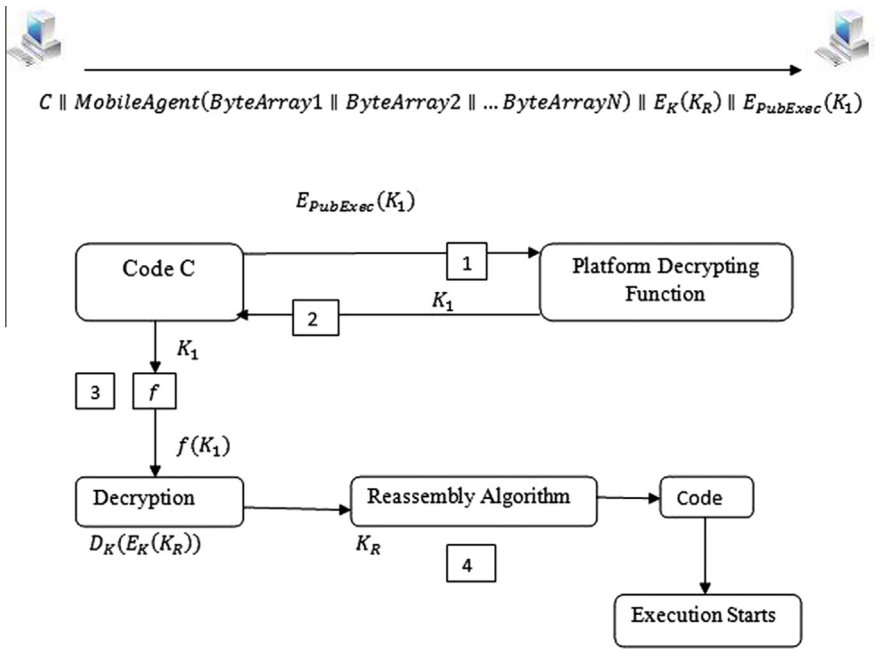



**Figure 1** Proposed security protocol.

Mobile Agent $\boldsymbol{M}$ = (byteArray1||byteArray2||byteArray3 … byteArrayN.) $K_S$

2. Embed this mobile agent into controller agent (code carrying agent $C$)
3. Controller agent $C$ has following program modules:
   Controller agent $C = Module1||Module2||Module3$
   $Module1$ = Platform interaction module
   $Module2$ = Self decryption module
   $Module3$ = Code reassembly and Execution module
4. $Module1$ communicates with execution platform or static agent which resides on that platform and sends request in the form of $E_{PubExec}$ $(K_1)$.
5. Static agent resides on that platform processes this request and decrypts it and sends this response $(K_1)$ to the requested module. $D_{PriExec}(E_{PubExec}(K_1)) = K_1$
6. $Module2$ processes on this key $K_1$ and generates symmetric decryption key $K$ which is a function of $K_1$, i.e. $K = f(K_1)$.
   $Module2$ has a key generation function $f$
7. Controller agent has one more Module 3 which has encrypted reassembly key $K_R$ (same as scattering key $(K_S)$ and algorithm.
   Module 3 = $E_K(K_R)$ || Reassembly Algorithm
8. After generating key $K$, module 2 decrypts encrypted reassembly key $K_R$ and reassembly algorithm and processes this module.
9. $Module3$ has reassembly algorithm, which reassembles the code from different byte arrays at run time and executes it.

### *3.2. Security and computational advantages over other techniques*

– In our protocol, whole code is not encrypted in order to reduce overhead. Here, only reassemble key is encrypted. As size of key is less as compared to whole code so we can reduce encryption overhead and decryption overhead.
– Our protocol obfuscates the code to provide privacy to the code as it is not encrypted but this obfuscation is slightly different as de obfuscation is performed by controller code itself at run time. So the agent, on communication channel is secured.
– If we compare our protocol with other protocol like other obfuscation technique. In obfuscation approach, agent owner do not sure whether his code can be de- obfuscated or not, means agent owner can prevent his code from being analyzed but cannot detect whether it has been analyzed or not.
– According to self protection technique, agent itself checks the security. In self protection scheme, there is a concept of public decryption function. In this technique, an interface works between the agent and the execution platform, interface is responsible for decrypting the agent code. Actually interface takes the private key from execution platform and decrypts agent' code which is encrypted with public key of execution platform. Before giving the private key to the interface, first the agent platform needs to authenticate the requested entity henceforth privacy of private key is not maintained over here.

Our protocol solves above stated problems, we also follow agent driven security, but here we secure the privacy of private key of the execution platform. In our protocol, agent itself calculates the decryption key $K$ which is a function of key $K_1$, i.e. $f(K_1)$ and after calculating decryption key $K$, it decrypts the encrypted reassembly key $K_R$ and assembles the code and executes it. In our protocol, the agent is less interactive with the execution platform which makes it secure form malicious behavior of the execution platform. Besides all these, our protocol also ensures the authenticity of entities involved in communication.

– Agent owner ensures the authenticity of the execution platform by sending key component $K_1$ to the execution platform by encrypting it with the public key of the next host.

– At execution platform, execution platform decrypts $K_1$ with its private key and gives it to the agent. In this way the agent ensures the authenticity of execution platform.

Agent takes this key $K_1$ and generates reassembly key which is the function of $f(K_1)$ at run time without any intervention of execution platform. This process ensures the execution privacy of code.

### 3.3. Proposed fragmentation based encryption approach

Till now, various techniques are proposed for securing mobile agent and its data from malicious entity, i.e. malicious execution environment or man in the middle attack, etc. In general AES is widely used for the encryption of large size data from the year 2001; whereas RSA is used for encrypting and distributing symmetric key (AES) whose size is very less as compared to the data.

In our protocol, the agent size is very less as compared to the data which may be encrypted with AES encryption but for this, we proposed a fragmentation based encryption technique where whole mobile code is not encrypted with AES rather only fragmentation and reassembly key is encrypted with AES key. Fragmentation and reassembly key is same like symmetric key. It is 128 bit long key in which each digit specifies the byte array order like.

128 bit means 16 byte key in which we can represent a single character (Special character also whose ASCII values lie between (0–255) with a byte e.g. key = 7 cd%&*@H3#cs9n$2.

#### 3.3.1. Fragmentation algorithm

There are 4 byte arrays of linked list where code is placed after fragmentation according to key. The 4 byte arrays of linked list are declared as follows. As ASCII value of character lies under the range of 0 to 255 and after applying mod 4 operations on these values, quotients will be found from 0 to 63.

In order to send the code through channel, the fragmentation algorithm breaks the code in different four byte arrays of linked list according to fragmentation algorithm 1. (Refer Algorithm 1 and Fig. 2.) First takes the bytes of codes one by one and places it into byte arrays of link list. Initially fragmentation algorithm divides the whole byte codes in different blocks of 16 bytes. Take first byte of first block and first special character of key. Apply modulus 4 operation on ASCII value of key like

Key = 7 cd%&*@ α∑ #cs9n$2

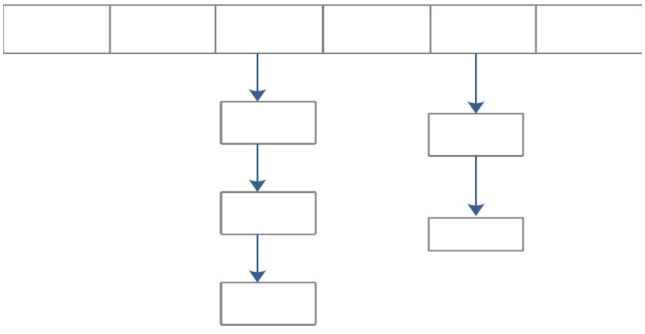


**Figure 2** Array of linked list.

*ASCII value of 7 = 55*

55 mod4 = 3 and quotient = 13 means first byte will be placed on byte array 3 at indexes 13. From the Table 1, we can see that character c occurs two times in key so that byte 2 and byte 11 of block are placed at array 3 and index of array will be 24. For that we used two dimensional arrays. Table 1 shows the allocation of a single block of 16 bytes. In the similar manner, all 16 bytes blocks of byte are placed.

In short suppose code size = 1 kb then

1Kbyte = 1024bytes = 64 blocks of 16 bytes
Key size = 128 bits = 16byte
So each block of 16 bytes is fragmented according to 16 bytes key. (Refer Fig. 3 and Algorithm 1.)

***Algorithm 1: Pseudo code for fragmentation algorithm***
***Inputs: Key*** $K_i$ **i: 1 → 16**
***//i.e.*** $K_1, K_2, K_3 \ldots K_{16}$ ***//Fragmentation Key***
***Data Block*** $b_i$ ***i: 1 → 16*** ***//i.e.*** $b_1, b_2 \ldots b_{16}$
***Outputs:***
***struct node***
{
***byte data;***
***struct node****$A_r$**[63];** // $r$:0 → 3 //i.e. Total 4 arrays of linked list.
}
***node****$A_r$**[63]** // Arrays of 63 node pointers.
***Begin:***
***for (i = 1 → 16)***
{
$K_i$ %4 → $r$;
$K_i$/4 → $q$;
$A_r[q]$ → ***data*** = $b_i$;
$A_r[q]$ → ***next*** = ***null***;
$i^{++}$;
}
***End***

***Algorithm 2***: ***Pseudo code for reassembly algorithm***
***Inputs***: *Key*$K_i$ ***i***: 1 → 16
//i.e. $K_1, K_2, K_3 \ldots K_{16}$//***Reassembly Key***
***struct node****$A_r$ [63]; // $r$:0 → 3, i.e. total 4 arrays of linked list.
***Outputs***: ***Data Block*** $b_i$***i***:1 → 16 ***//i.e..*** $b_1, b_2 \ldots b_{16}$
***Begin:***
***for(i = 1to16)***
{
$K_i$%4 ← $r$;
$k_1$/4 ← $q$;
$b_i$ = $A_r$ - $r[q]$ → ***data***;
$i^{++}$;
}
***End***

In our approach, the key (Fragmentation Key) is encrypted with AES key and sent with mobile agent. On the receiver side, code is assembled according to the key after decrypting the key with AES key according to reassembly algorithm 2. Actually

**Table 1** Fragmentation and ordering of bytes of code.

| Byte of code | Key character | ASCII value | Mod4 (Byte Array No.) | Quotient (Index) |
|---|---|---|---|---|
| Byte 1 | 7 | 55 | 3 | 13 |
| Byte 2 | C | 99 | 3 | 24 |
| Byte 3 | D | 100 | 0 | 25 |
| Byte 4 | % | 37 | 1 | 9 |
| Byte 5 | & | 38 | 2 | 9 |
| Byte 6 | * | 42 | 2 | 10 |
| Byte 7 | @ | 64 | 0 | 16 |
| Byte 8 | A | 224 | 0 | 56 |
| Byte 9 | ∑ | 228 | 0 | 57 |
| Byte 10 | # | 35 | 3 | 8 |
| Byte 11 | C | 99 | 3 | 24 |
| Byte 12 | s | 115 | 3 | 28 |
| Byte 13 | 9 | 57 | 1 | 14 |
| Byte 14 | N | 110 | 2 | 27 |
| Byte 15 | $ | 36 | 0 | 9 |
| Byte 16 | 2 | 50 | 2 | 12 |

the execution platform is not able to directly decrypt the fragmentation key because it fails to calculate the symmetric AES key. Execution platform gets the encrypted key $K_1$. Platform decrypts the encrypted component with its public key and gives it to the execution code module 3 which calculates AES key $K$ which is the function of the key $K_1$, i.e. $f(K_1)$. Now agent decrypts the reassembly key itself and assembles the code according to the reassembly key. Fig. 4 demonstrates the whole protocol based on fragmentation based encryption.

## 4. Formalization of our security protocol

### 4.1. Security threat classification

Focusing on the security aspects, the following are identified as the comprehensive security threats associated to the mobile agent technology. In a nutshell, there are two major aspects of protection mechanism. Before delving into the details of threat classification, first we classify the entities involved in attack.

- Protection of execution environment.
- Protection of mobile agent.

But here, we classified attacks only related to agent security. In other words, agent security means protecting agent from execution platform or from MITM (channel attacker or man in the middle attack). In order to classify attacks, we need to classify the attackers, agent's components which are vulnerable to attack and the security factors (confidentiality, integrity and availability) which can be compromised.

#### 4.1.1. Formal representation of attacks

As discussed in previous section, researchers have provided various solutions for the protection of agent platform which is broadly acceptable but the protection of mobile agent still a concern area for research. We formalize the security threats and attacks related to agent security in following way.

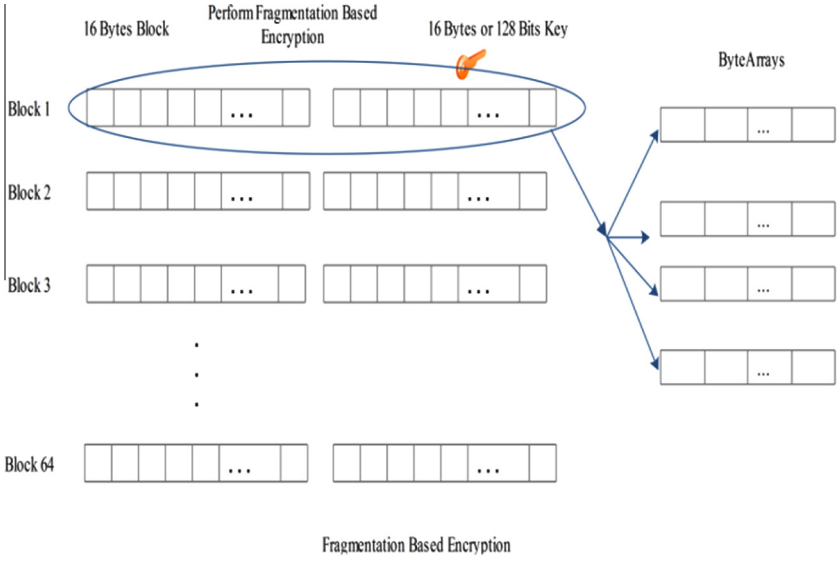



**Figure 3** Fragmentation of 1 Kbyte of code.

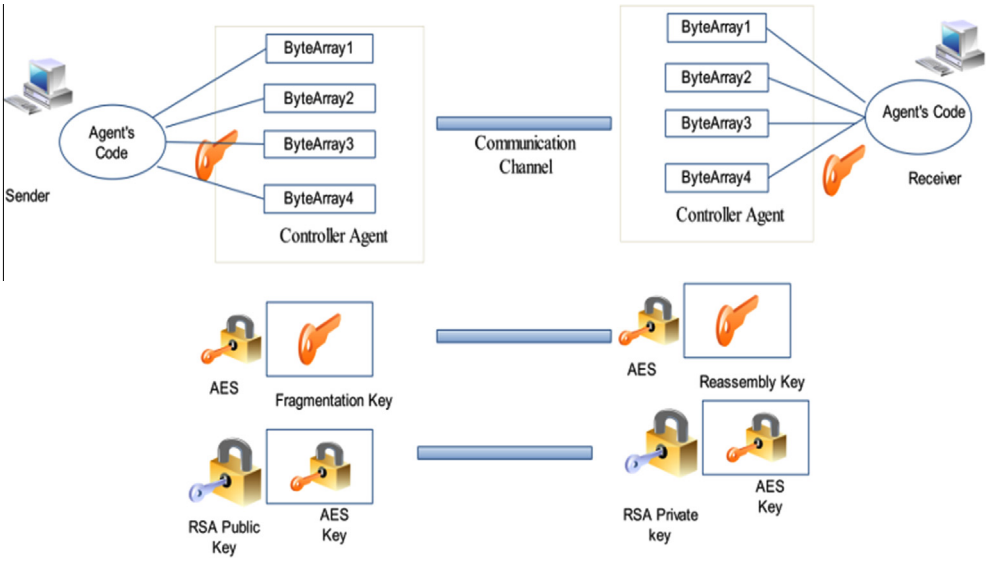



**Figure 4** Fragmentation based encryption.

**Table 2** Formal classification of attacks.

| $\{A, M, M_C\}$ attacks on agent's code | | $\{A, M, M_S\}$ attacks on agent's state | | $\{A, M, M_D\}$ attacks on agent's data | |
|---|---|---|---|---|---|
| Security attacks classification $\{A, M, MC, F\}$ | | | | | |
| Security attacks classification $\{A, M, MC, F\}$ | | | | | |
| $\{CA, M, M_C\}$ | $\{MH, M, M_C\}$ | $\{CA, M, M_S\}$ | $\{MH, M, M_S\}$ | $\{CA, M, M_D\}$ | $\{MH, M, M_D\}$ |
| $\{CA, M, M_C, C\}$ | $\{MH, M, M_C, C\}$ | $\{CA, M, M_S, C\}$ | $\{MH, M, M_S, C\}$ | $\{CA, M, M_D, C\}$ | $\{MH, M, M_D, C\}$ |
| $\{CA, M, M_C, I\}$ | $\{MH, M, M_C, I\}$ | $\{CA, M, M_S, I\}$ | $\{MH, M, M_S, I\}$ | $\{CA, M, M_D, I\}$ | $\{MH, M, M_D, I\}$ |
| $\{CA, M, M_C, A\}$ | $\{MH, M, M_C, A\}$ | $\{CA, M, M_S, A\}$ | $\{MH, M, M_S, A\}$ | $\{CA, M, M_D, A\}$ | $\{MH, M, M_D, A\}$ |

**Table 3** Notation used in attack classification.

| Symbols | Definitions |
|---|---|
| CA | Channel attacker |
| MH | Malicious host |
| MC | Compromised Agent's component |
| $M_C$ | Agent's code component |
| $M_D$ | Agent's data component |
| $M_S$ | Agent's state component |
| $C$ | Confidentiality |
| $I$ | Integrity |
| $A$ | Availability |

*4.1.2. Attack scenario*

We formalize the attack in this respect.

Attack = $\{A,M,MC,F\}$ *means "attacker A performs attacks on mobile agent M and affected component of mobile agent is 'MC' and factor 'F' of security is compromised.*" (Refer Table 3 for notation.)

where Attacker A can be classified as = $\{CA, MH\}$ = {Channel Attacker, Malicious host} A set of attacks may be possible in agent scenario mentioned in Table 2.

It can be seen from the above formalism there are 24 classes of attacks. In our approach, we will focus on the data and code confidentiality and integrity issues. Hence, according to the formalism following 8 classes of security threats will be focused.

$$\{CA, M, \mathrm{M_C}\}, \{CA, M, \mathrm{M_C}, C\}, \{CA, M, \mathrm{M_C}, I\},$$
$$\{CA, M, \mathrm{M_C}, A\}, \{MH, M, M_C\}, \{MH, M, M_C, C\},$$
$$\{MH, M, M_C, I\}, \{MH, M, M_C, A\}$$

So from a single approach we will solve 33.33% security threats. Shaded cell of Table 2 represents the security objectives solved by our protocol. In the Section 3.2, we will show how our protocol fulfils the above mentioned security requirements.

We can verify our security protocol formally as follows. Suppose Mobile agent $M$ is created by agent owner platform $OP$ and it is transferred from owner platform to platform $P_i$ with the help of controller agent $C$ which is a code (agent) carrying agent (Ma, 2006; Ma and Tsai, 2008). Table 4 represents the meaning of formal representations.

1. Owner platform creates the mobile agent and sends it to platform $P_i$ and security related aspects.

$$(OP, f_{frag}(M)_{K_S} \| C(M_1 \| M_2 \| M_3) \| f_{senc}(K_S)_K \| f_{Aenc}(K_1)_{Pub_i}\}, P_i)$$

2. Agent platform $P_i$ receives the agent and processes it.

$$(P_i, \mathrm{OP}, \{f_{frag}(M)_{K_S} \| \mathrm{C}(M_1 \| M_2 \| M_3) \| f_{Senc}(K_S)_K \| f_{Aenc}(K_1)_{Pub_i}\})$$

**Table 4** Representation of formal's parameters.

| Formalism | Meaning |
|---|---|
| $OP$ | Agent owner platform |
| $P_i$ | Agent platform |
| $M$ | Mobile agent |
| $C$ | Controller agent |
| $M_1$ | Platform interaction module |
| $M_2$ | Self decryption module |
| $M_3$ | Code reassembly and execution module |
| $(A, \{X\}, B)$ | Entity $A$ sends $X$ to entity $B$ |
| $(B, A, \{X\})$ | Entity $B$ receives $X$ from entity $A$ |
| $f_{frag}(M)_{K_S}$ | Fragments the code in different byte array according to key $K_S$ |
| $f_{Senc}(K_S)_K$ | Performs symmetric encryption by symmetric key $K$ |
| $f_{Aenc}(K_1)_{Pub_i}$ | Performs asymmetric encryption with public key of $i$th host |
| $f_{Adec}(K_1)_{Pri_i}$ | Performs asymmetric decryption with private key of $i$th host |
| $f_{Sdec}(K_S)_K$ | Performs symmetric decryption by symmetric key $K$ |

3. As we previously described in the protocol that controller agent enfolds three modules for specific purposes. Here platform interaction module $M_1$ interacts with platform $P_i$ as follows. Platform $P_i$ performs decryption operation and sends key $K$ to platform interaction module $M_1$.

$$(P_i, \{f_{Adec}(K_1)_{Pri_i}\}, M_1) = (P_i, K_1, M_1)$$

4. Self decryption module $M_2$ processes on this key $K_1$ and generates symmetric decryption key $K$ which is a function of $K_1$. Module $M_2$ has a key generation function $f$.

$$(M_2, \{f_{kgen}(K_1)\}, \text{process})$$

5. Self decrypting module $M_2$ decrypts scattering key $K_S$ which also works as a reassembly key and gives this key to code reassembly and execution module $M_3$.

$$(M_2, \{f_{Sdec}(K_S)_K\}, M_3) = (M_2, K_S, M_3)$$

6. Execution module $M_3$ reassembles the code and executes it.

$$(M_3, \{f_{Rass}(M)_{K_S}\}, \text{process})$$

### 4.2. Formal verification using BAN logic

Security protocols are used for any number of intended purposes like, authentication, confidentiality and non-repudiations, etc. In this part of formal verification (BAN logic), we will focus on authenticated establishment of session keys and formally verify the protocol through BAN logic. In this section, we will not go into the detail of BAN logic. The language of BAN logic which is used in our self protection protocol is as follows:

1. $A$ *believe* $B$: $A$ may act as if $B$ is true.
2. $A \overset{K}{\leftrightarrow} B$ : '$K$' is a good shared key for $A$ and $B$ means $K$ will never be discovered by any principal but $A$, $B$ or a principal trusted by $A$ and $B$.
3. $\overset{K}{\leftrightarrow} A$ :$K$ is the public key of principal $A$. $K^{-1}$ behaves as corresponding private key that will never be discovered by any principal but $A$.
4. $\{X\}_K$: Read $X$ encrypted with key $K$, $K$ may be either symmetric key or asymmetric key.
5. $A \triangleleft X$: Principal $A$ received or see $X$.

However, there are more notations and rules exist in BAN logic, but all those are out of the scope of our protocol's verification.

**Initial state assumption**
**P1**: $OP$ *believes* $OP \overset{K_i}{\leftrightarrow} P_i$
**P2**: $OP$ *believes* $\overset{Pub_i}{\rightarrow} P_i$
**P3**: $P_i$ *believes* $P_i \overset{K_1}{\leftrightarrow} M_1$
**P4**: $M_1$ *believes* $M_1 \overset{K_1}{\underset{K_S}{\leftrightarrow}} M_2$
**P5**: $M_2$ *believes* $M_2 \overset{K_S}{\leftrightarrow} M_3$

Initially agent's owner platform $OP$ creates the mobile agent $M$ and sends it to platform $P_i$ and security related aspects. In Message 2, we have shown that the key component $K_1$ is shared between platform $P_i$ and $M_1$. In Message 3, we have shown that, platform interaction module $M_1$ gives $K_1$ to self decryption module $M_2$ from the Message 4, it can be seen that, fragmentation/reassembly key $K_S$ is only be shared by module $M_2$ and module $M_3$. There is no interference of execution platform over fragmentation and reassembly key as well as symmetric key $K$.

**Idealized form of self protection protocol**
**Message 1**:
$OP \rightarrow P_i : \{f_{frag}(M)_{K_S}, C(M_1 \| M_2 \| M_3), \{K_S\}_K, \{K_1\}_{pub_i}\}$
**Message 2**: $P_i \rightarrow M_i : \{P_i \overset{K_1}{\rightarrow} M_1\}$
**Message 3**: $M_1 \rightarrow M_2$:$\{M_1 \rightarrow M_2\}$
**Message 4**: $M_2 \rightarrow M_3$:$\{M_2 \rightarrow M_3\}$

After protocol idealization, we stated the initial state assumptions. All the assumptions are related to share keys between principal's components. Here principals mean agent platform, agent's owner, agent's module ($M_1$, $M_2$, $M_3$). $P_1$ tells the belief of OP, i.e. $K_1$ is shared secret key between agent's owner platform $OP$ and agent platform $P_i$. $P_2$ states the belief of $OP$ that $P_i$ has a public key $Pub_i$. $P_3$ through $P_5$ tells us the belief of principals $P_i$, $M_1$ and $M_2$ for their shared secret keys $K_1$, $K_2$ and $K_S$.

We fetched the inferences mentioned in conclusion part of formal verification. $C1$ tells that owner platform $OP$ ensured that symmetric key's component $K_1$ is only accessed by principal $P_i$ (Agent's platform) as it is encrypted with the public key of $P_i$, i.e. $Pub_i$. $C2$ stated the belief of agent owner $OP$ that module $M_2$ has key generation function $f_{kgen}$ to calculate shared symmetric key $K$. $C3$ tells us that fragmentation/reassembly key $K_S$ is shared between $M_2$ and $M_3$.

**Conclusions**
***C*1**: $OP$ *believes* $\overset{Pub_i}{\rightarrow} P_i,\ P_i \triangleleft \{K_1\}_{Pub_i}$
***C*2**: $OP$ *believes* $M_1$ *believes* $M_1 \overset{K_1}{\underset{K_S}{\leftrightarrow}} M_2,\ M_2 \triangleleft f_{kgen}(K_1), M_2 \triangleleft K$
***C*3**: $OP$ *believes* $M_2$ *believes* $M_2 \overset{K_S}{\leftrightarrow} M_3,\ M_2 \triangleleft \{K_S\}_K,\ M_2 \triangleleft K_S$

### 4.3. Security analysis on the basis of formalism

In initial step 1, agent owner first creates controller agent comprises three modules which is dependent with one another.

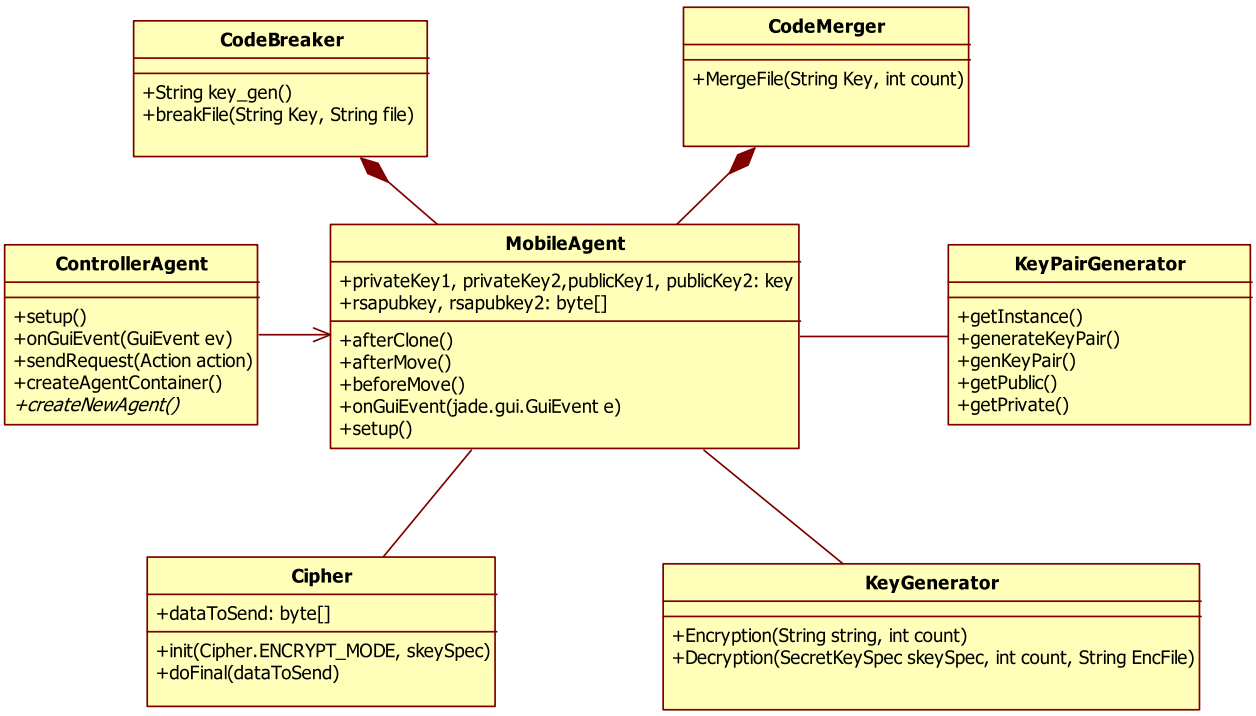



**Figure 5** Class diagram of implemented protocol.

**Table 5** Notation representation of timing analysis.

| Notation | Meaning |
|---|---|
| $T_{E,AES}$ | Time taken by AES encryption to encrypt mobile code |
| $T_{D,AES}$ | Time taken by AES decryption to decrypt mobile code |
| $T_{E,RSA}$ | Time taken by RSA encryption |
| $T_{D,RSA}$ | Time taken by RSA decryption |
| $T_{E',AES}$ | Time taken by AES encryption to encrypt fragmentation key |
| $T_{D',AES}$ | Time taken by AES decryption to decrypt fragmentation key |
| $T_f$ | Code fragmentation time |
| $T_R$ | Code reassembly time |

Controller agent carries mobile agent as a data variable. In order to provide security, agent owner performs fragmentation based encryption. Fragmentation based encryption is basically a combination of encryption as well as obfuscation. Encryption provides confidentiality whereas obfuscation makes the code difficult to understand. Looking from the attacker's perspective, attacker may perform brute force attack to reassemble the code. Following are the possible attempts to compromise security.

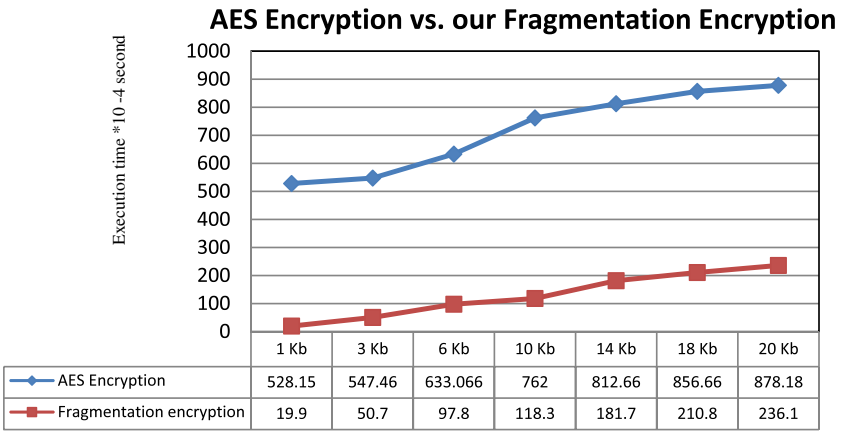
- Initially attacker may be confused about the whole mobile agent structure because it is fragmented in different byte arrays.
- Suppose in any ways, attacker comes to know that agent is in the form of fragmented codes then it will try to reassemble it.
- As fragmentation executes according to the key specified by agent creator, so attacker may perform the brute force attack, i.e. chosen key attack, but it is a challenge as the code is scattered in different parts and some dead code is also inserted in these part to make it hard to crack.
- The one most important part of our protocol is that, here agent owner encrypts the fragmentation key with AES, so attacker gets the encrypted fragmentation key, so first attacker will apply the brute force attack to break the AES security which is also secured with RSA in our protocol.
- All above points are mentioned for attackers which may be any malicious entity like malicious agent, malicious host or man in the middle attacker.

But for the case, where execution platform of agent itself try to perform malicious activity is the main challenge which still exists as a problematic concern hence our protocol proposed a self decryption module to override this issue.

In traditional approach AES key is distributed securely through RSA key distribution approach but for the malicious execution problem, we cannot provide AES key to the execution platform to directly decrypt the code and execute it. Our protocol only distributes the key component of AES key to the execution platform. Execution platform decrypts AES key component and gives it to the intended self decryption module. This process ensures the authenticity of execution platform as only this platform has the private key to decrypt the encrypted AES key components.

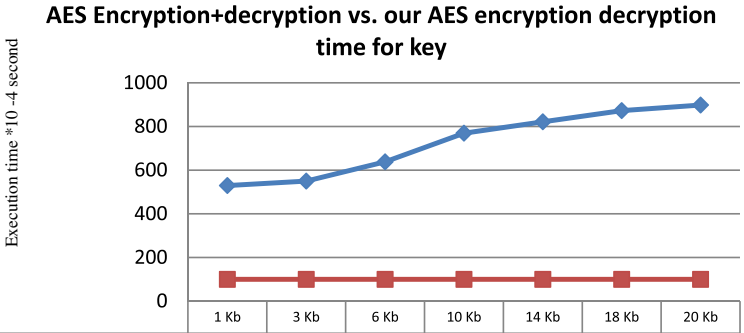
## 5. Implementation details

The whole scheme has been implemented and tested as an add-onto the well-known JADE platform, providing a down-to-earth realization of the proposed mechanisms. This



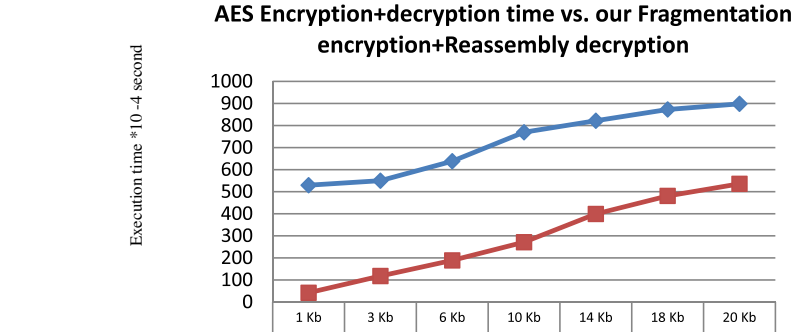

| | 1 Kb | 3 Kb | 6 Kb | 10 Kb | 14 Kb | 18 Kb | 20 Kb |
|---|---|---|---|---|---|---|---|
| AES Encryption | 528.15 | 547.46 | 633.066 | 762 | 812.66 | 856.66 | 878.18 |
| Fragmentation encryption | 19.9 | 50.7 | 97.8 | 118.3 | 181.7 | 210.8 | 236.1 |

**Figure 6** Execution time analysis of AES and fragmentation encryption.




| | 1 Kb | 3 Kb | 6 Kb | 10 Kb | 14 Kb | 18 Kb | 20 Kb |
|---|---|---|---|---|---|---|---|
| AES Encryption+decryption time | 529.53 | 549.92 | 638.21 | 769.47 | 821.62 | 872.8 | 898.36 |
| AES encryption decryption time for key | 100 | 100 | 100 | 100 | 100 | 100 | 100 |

**Figure 7** Comparisons of AES execution times.




| | 1 Kb | 3 Kb | 6 Kb | 10 Kb | 14 Kb | 18 Kb | 20 Kb |
|---|---|---|---|---|---|---|---|
| AES Encryption+decryption time | 529.53 | 549.92 | 638.21 | 769.47 | 821.62 | 872.8 | 898.36 |
| Fragmentation encryption+Reassembly decryption | 42 | 117.9 | 188.7 | 271.2 | 399.2 | 480.9 | 535.7 |

Figure 8. Comparisons of AES Encryption with Fragmentation Encryption

**Figure 8** Comparisons of AES encryption with fragmentation encryption.

implementation fosters our confidence in the viability of the protection scheme. JADE (Java Agent Development Framework) (Vila et al., 2007; Board, 2005; Bellifemine et al., 2010; Caire, 2009) is a software development framework aimed at developing multi-agent systems and applications conforming to FIPA standards for intelligent agents. JADE is written in Java language and is made of various Java packages, giving application programmers both ready-made pieces of functionality and abstract interfaces for custom, application dependent tasks.

We tested our application on a LAN with a server act as an agent owner and three clients exist as an execution environment for the agent. In our proposed model a main container has to be formed at the server's side which would be

containing the entire containers present in the client's sides of network. We suppose each container as a host and main container as an agent owner for the simulation of our protocol.

We create a controller agent that triggers the movement of the mobile agent from the origin. Controller Agent sends the request to the Mobile Agent to move itself through Agent Communication language (ACL). Code and Data are encapsulated in a mobile agent passes through host to host. Code performs the task of retrieving the data of the particular host it visits, and stores it in the variable.

In our algorithm, code is a java byte code which is written in Executable.class file. According to the Algorithm 1 stated earlier, fragments Executable.class file into different byte arrays (arrays of linked list) with the help of CodeBreaker.class and encapsulates it in the mobile agent. For providing mobility, mobile agent performs class serialization and assembles the mobile code at receiver end with CodeMerger.class. Code Breaker function performs fragmentation according to the 128 bit key which is distributed securely through AES and RSA encryption techniques.

We have implemented our protocol using java and its security library. In order to implement whole protocol we write following classes. Each class has its own specific purpose. (Refer Fig. 5.)

- Code Breaker
- Code Merger
- KeyPair Generator
- Key Generator
- Controller Agent
- Mobile Agent
- Cipher

In general, in case of mobile agent scenario, whole code is encrypted with AES key and then sends this key to the receiver end in encrypted manner using RSA public key. The whole formal representation of this process is:

$$E_{K,AES}(M) + E_{pub,RSA}(K) \tag{1}$$

But in our case, we break the mobile agent's code in different byte array according to fragmentation key $K_f$. We apply AES encryption on this fragmentation key of size 128 bit long. As size of fragmentation key is smaller than the whole mobile agent' code so it creates less encryption overhead.

The formal representation of this process is:

$$E_{K,AES}(K_f) + E_{pub,RSA}(K) + E_{K_f}(M) \tag{2}$$

### 5.1. Time complexity analysis

In order to model the execution time of traditional approach and our approach, we assume the following mathematical representation which is shown below in Table 5.

In mobile agent scenario, each host in a network behaves like as sender as well as receiver, so the total time taken by traditional approach is

$$= T_{E,AES} + T_{D,AES} + T_{E,RSA} + T_{D,RSA} \tag{3}$$

And total time taken by our security protocol

$$= T_{E',AES} + T_{D',AES} + T_f + T_R + T_{E,RSA} + T_{D,RSA} \tag{4}$$

Comparing both Eqs. (3) and (4), it seems that traditional approach is better but in our case (Mobile agent case) where code size is less (1 kb to 20 kb) our protocol behaves better as we previously said instead of encrypting whole code, we encrypt only fragmentation or reassembling key which is of 16 bit only that is very small as compared to code size. So the total time taken by AES encryption and decryption is less in our case as compare to traditional protocol which can be shown by inequalities (1) and (2).

$$T_{E',AES} \ll T_{E,AES} \tag{inequ1}$$

$$T_{D',AES} \ll T_{D,AES} \tag{inequ2}$$

In Eqs. (3) and (4) RSA execution time is common, but if we apply inequalities comparison then we found that our protocol is better as compared to traditional approach.

$$T_{E',AES} + T_f < T_{E,AES} \quad \text{And} \tag{inequ3}$$

$$T_{E',AES} + T_f + T_R + T_{D',AES} \ll T_{E,AES} + T_{D,AES} \tag{inequ4}$$

### 5.2. Testing and results

First we take the AES encryption and decryption time on 1 kb to 20 kb class file size. Our experimental observations are shown in Figs. 6–8. For each class file, we take 15 execution sample and then apply averaging operation on it. In the same way we take fragmentation and reassembly computational time for each class file.

From the Table 5, we can observe,

$$T_{E',AES} + T_f + T_R + T_{D',AES} \ll T_{E,AES} + T_{D,AES} \tag{inequ5}$$

$$\text{and} \quad T_{E',AES} + T_f < T_{E,AES} \quad \text{are true in our case.} \tag{inequ6}$$

The whole operation is implemented and tested with the configuration of 2.2 GHz dual core processor and 4 GB RAM.

Fig. 6 represents the comparison between AES encryption and fragmentation encryption. As agent size is very much less as compared to the data size, encryption time of fragmentation encryption is less as compared to AES encryption. We are not challenging the AES algorithm strength for large size data but in mobile agent case, our fragmentation based approach behaves better.

Fig. 7 gives the inference that our approach's performance is immutable as the code size increases because in our case, key is encrypted by AES while in traditional approach, whole code is encrypted whose size is more as compared to the fragmentation or reassembly key.

Fig. 8 shows the comparison between AES based encryption decryption and our fragmentation based encryption and decryption. In Fig. 8 it can be seen for small size of agent's code our technology takes less computational time but as the size of code increases our algorithm graph's line moves towards AES's graph line.

## 6. Conclusions and future works

In this paper, we gave a new direction to traditional approach which makes the process of encryption more efficient and takes less computational time in agent scenario where code moves towards data location rather than data. As agent's code is

small in size which reduces network overhead and solves the problem of bandwidth and network latency. So our main motive is to provide desirable security to agent based infrastructure at a very less cost in term of time and space.

To accomplish all these stuff, we proposed a security protocol which is a combination of self decryption, co-operation and obfuscation technique also inbuilt with proposed fragmentation based encryption to afford security with mitigated computational cost. Our fragmentation based encryption made encryption technique harder and provides thorny obfuscation for attacker. Since in general mobile agent size is very much less that is suited for our fragmentation based approach which behaves better than traditional AES and RSA based encryption.

Future agenda and scope of this technique is to incorporate this technique for sending short critical data in network. To achieve this we will have to focus on the complexity part of the algorithm.

## References


Ahmed, Tarig Mohamed, 2009. Using secure-image mechanism to protect mobile agent against malicious hosts. World Academy of Science, Engineering and Technology.

Ametller, J., Robles, S., Ortega- Ruiz, J.A., 2004. Self-Protected Mobile Agents. ACM, New York, pp. 362–367.

Armoogum, Sandhya, Caully, Asvin, 2011. Obfuscation techniques for mobile agent code confidentiality. Journal of Information & Systems Management, 25–36.

Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimassa, JADE Programmer's Guide, April 8, 2010. JADE 4.0.

Benachenhou, Lotfi, Pierre, Samuel, 2006. Protection of a mobile agent with a reference clone. Computer Communications, vol. 29. Elsevier, pp. 268–278.

Borselius, N., 2002. Mobile agent security. Electronics & Communication Engineering Journal, 211–218.

Bouchemal, Narjes, Maamri, Ramdane, Sahnoun, Zaidi, 2009. CAP: clone agent protocol to protect mobile agents. MASAUM Journal of Basic and Applied Sciences 1 (1).

Giovanni Caire, JADE Programming for Beginners, June 30, 2009. JADE 3.7.

El Rhazi, Abdelmorhit, Pierre, Samuel, Boucheneb, Hanifa, 2007. A secure protocol based on a sedentary agent for mobile agent environments. Journal of Computer Science 3 (1), 35–42.

Hohl, Fritz, 1998. Time limited black box security: protecting mobile agents from malicious hosts. In: Vigna, G. (Ed.), Mobile Agents and Security, Lecture Notes in Computer Science, vol. 1419. Springer-Verlag, Berlin, pp. 92–113.

Guan, Huanmei, Zhang, Huanguo, 2010. A communication security protocol of mobile agent system, vol. 15 (No. 2). Wuhan University and Springer-Verlag, Berlin, Heidelberg, pp. 117–120.

JADE Board, JADE Security Guide, February 28, 2005. JADE 3.3.

W.A. Jansen, T. Karygiannis, 2000. Mobile agent security, NIST: National Institute of Standards and Technology, Special Publications 800-19.

Lange, D.B., Oshima, M., 1999. Seven good reasons for mobile agents. Communications of the ACM 42 (3), 88–89.

Hyungjick Lee, 2004. The use of encrypted functions for mobile agent security. In: Proceedings of the 37th Hawaii International Conference of IEEE on System Sciences.

Hyungjick Lee, Jim Alves-Foss, 2004. The use of encrypted functions for mobile agent security. In: Proceedings of the 37th Hawaii International Conference on System Sciences, pp. 1–10.

Ma, Lu., Tsai, Jeffrey J.P., 2006. Security Modeling and Analysis of Mobile Agent System. Imperial College Press, 57 Shelton Street, Covent Garden, London WC2H 9HE.

Ma, Lu, Tsai, Jeffrey J.P., 2008. Formal modeling and analysis of a secure mobile-agent system. IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans 38 (1).

Nathan Macrides, 2002. Security techniques for mobile code, SANS Security Essentials (GSEC) Practical Assignment Version 1.4.

Ouardani, Abdelhamid, Pierre, Samuel, Boucheneb, Hanifa, 2007. A security protocol for mobile agents based upon the cooperation of sedentary agents. Journal of Network and Computer Applications 30, 1228–1243.

Roth, V., 1998. Secure recording of itineraries through co-operating agents. In: Demeyer, S., Bosch, J. (Eds.), ECOOP Workshops, Lecture Notes in Computer Science, vol. 1543. Springer, Brussels, Belgium, pp. 297–298.

Roth, V., 1999. Mutual protection of cooperating agents. In: Vitek, J., Jensen, C. (Eds.), Secure Internet Programming: Security Issues for Mobile and Distributed Objects. Springer Verlag.

Sander, T., Tschudin, C.F., 1998. Protecting mobile agents against malicious hosts. In: Vigna, G. (Ed.), Mobile Agents and Security, Lecture Notes in Computer Science, vol. 1419. Springer, pp. 44–60.

Shashank Srivastava, G.C. Nandi, 2011a. Detection of mobile agent's blocking in secure layered architecture. In: 2011IEEE International Conference on Communication Systems and Network Technologies, Katra-Jammu, India, pp. 43–48.

Shashank Srivastava, G.C. Nandi, 2011b. Protection of mobile agent and its itinerary from malicious host. In: Second IEEE International Conference on Computer & Communication Technology (ICCCT)-2011, Allahabad, India, pp. 405–411.

Vila, X., Schuster, A., Riera, A., 2007. Security for a multi-agent system based on JADE. Computers & Security 26 (3), 91–400.