A Comparative Study of Security Techniques for Protecting Mobile Agents from Malicious Hosts

Anupam Jain1, Kunwar Singh Vaisla2 and Zia Saquib3
1M.Tech CSE, Department of Computer Science and Engineering, BTKIT Dvarahat, Almora, Uttarakhand, India
2Associate Professor, Department of Computer Science and Engineering, BTKIT Dvarahat, Almora, Uttarakhand, India
3Executive Director, Centre for Development of Advanced Computing, Mumbai, India
e-mail: 1anupamjain2198@gmail.com, 2vaislaks@rediffmail.com, 3zsaquib@gmail.com

Abstract—In the modern world, mobile agent technology offers new innovating paradigm in which agent with the capability to migrate from one host to another where it can resume its execution. Mobile agent is the incarnation of highly organized software with embedded intelligence. Mobile agents are gaining the researcher’s attention due to its valuable features. Despite a number of successful mobile agent application still there are some barriers preventing this technology from spreading out to wider range of enterprise and individual user. Security issues have an important role in the development of secure and anti-tamper mobile agent system. This paper provides an overview of range of measures for countering the identified threats and fulfilling these security objectives.

Keywords: Mobile agent, Black box, Host, Mobility

I. INTRODUCTION

Mobile agent paradigm has many advantages over the traditional network computing models like reduce network traffic, overcome network latency. As the sophistication of mobile agent has increased over time, it has being the tremendous threats to security. Security aspects in the mobile agent system can be classified as the following four main categories: Agent to host, agent to agent, host to agent, and others to agent/host.

The issue of security hosts from malicious agents has been widely investigated and researched. Many mechanisms have been developed to protect the host from hostile agent. The mobile agent security against hostile hosts is one of the most crucial subjects in mobile agent technology. Sander and Tschudin present two types of security problems that must be solved [1]. The first is host protection against tampering agents. The second is agent protection against hostile hosts. Many mechanisms have been developed for the first kind of problem, such access control, password protection, sandboxes but second one problem is still being a challenging issue for researchers. Mobile agent can be described as follows:

A mobile agent is a software program that migrates from node to node of a heterogeneous network on the user’s behalf to perform the task [1][2]. They consist three parts: code, a data state, and an execution state. Code mobility is the main properties of mobile agent, which have capability to change the bindings between code fragments, and the location where they are executed dynamically [3]. There are two types of mobility level: weak mobility and strong mobility. In the weak mobility, mobile agent carries data state and code and on moving, the execution has to start from the beginning. In the strong mobility, mobile agent carries data state, code and execution state.

While in strong mobility, the execution can continue from the point it stopped on the pervious host. Mobile agents are goal oriented.

The main issue of security is safely executing the code of the mobile agent in the trusted environment [2]. Once the mobile code and data reaches to the host, they are fully under the control of the host mercy. Hosts have the capability to manipulate the code and data. They can brainwash the previous activities or collected information of mobile agent for their personal benefit.

Hostile hosts can do the following attacks:

1. Manipulation of code.
2. Manipulation of data.
3. Manipulation of interaction with other agents or hosts.
4. Eavesdrops on the code and data.
5. Eavesdrops on the execution of code.
6. Erase the previous information collected by agent.
7. Return the incorrect data.

One solution to tackle these security problems is that trusted nodes or hosts can restrict the agents. Limiting an agent to trusted node is limiting the usefulness of mobile agent paradigm. In today’s electronic world, one agent can visit finite number of hosts. In addition, how does an agent determine that a particular node is trusted or not and how will a trusted node will be add in the system. Because mobile agents execute on remote hosts and some experts think that, there is no way to ensure safety of an agent without using tamper-resistant hardware.

There is number of possible software solutions to protect mobile agents from malicious hosts. This paper will explore possible mobile agent applications, some of security problems associated with these applications, and after this, we will discuss some threats in mobile agent system and various mechanisms that have been proposed to protect the mobile agent system.
II. NEED OF SECURITY IN MOBILE AGENT APPLICATIONS

Let us discuss a classical example Fig. 1, which is using the mobile agent technology to determine the best price of airline ticket. In this example, the agent is sent out to find the best possible price for an airline ticket to a specific destination. An agent will be sent to visit each host that offers the airline tickets. The agent will return to the user when all of the possible ticket prices have been evaluated, and it will return with a list of best prices to the agent’s owner [24].

Several types of attacks are possible on the agent searching for airline tickets. If we do not provide any security to agent then possible types of attacks are listed below.

1. A malicious host can change the data that is carried by an agent for example a malicious host could delete airline ticket prices that are cheaper than it can offer, in an attempt to win the agent’s business.
2. If mobile agent has some form of e-cash then a malicious host can steal it.
3. A malicious host could modify the flow control of an agent so that the agent will bypass other hosts with cheaper airline tickets.
4. If agent’s code is in plain text then a malicious host can learn the algorithms used in implementation of agent and make some changes on agent’s code and it will never complete its task.

These types of attacks show the importance of security mechanism in mobile agent paradigm without applying the proper security we can’t use the mobile agent technology in e-commerce applications.

III. POSSIBLE THREATS TO MOBILE AGENT

In mobile agent system a malicious host is describe as a host that executes mobile agent and tries to make any kind of attack. When an agent executes on a host then it uses all the resources from the host and host is capable to monitor memory used by agent and each instructions given by the agent to the host.

Therefore, a malicious host may attempt attacks to agent in number of ways [25]:

- a. A host masquerading as another host.
- b. Denial of service by the host to the agent.
- c. Eavesdropping on an agent’s activity.
- d. Alteration of the agent by the host.

A. Masquerading

In masquerading, a host is try to make agent into believing that it is another host causing and the agent to give the host sensitive information. Once the masquerading host gains the trust on agent it may then be able to read or modify any of agent’s code. The solution to prevent this kind of attack is to use strong authentication protocol to authenticate host from malicious host. In the following figure describes how masquerading works. Here a host name Joe steals the login id and password of another host Sarah and by using these cardinals Joe pretending that he is Sarah. Now he will gain trust on any coming mobile agent that he is Sarah and access all sensitive information.

B. Denial of Services

In this kind of threat, a host may deny a mobile agent from its specific service. This type of attack is possible in two ways:

- 1. A host may deny an agent intentionally and unintentionally from its service and agent is not able to complete its task.
- 2. Another type of attack is host could terminate the agent altogether.

A host can also deny the request generated by an agent, in the case of time sensitive task agent is not able to complete its task in allotted time slot because agent’s request is denied by host.

In the Fig. 2, the attacker is a malicious host and create a lot of zombie request to Victim host(which is not malicious) due to lot of requests victim host starts denies the requests now if an mobile agent will submit a request then victim host will unintentionally deny this request, this kind of threats is called denial of services.

C. Eavesdropping

The next attack that can be performed by a malicious host is eavesdropping. This kind of attack is mostly performed in typical client/server model.
D. Alteration

The last attack that can be performed by a malicious host is alteration of code, data and control of an agent. Malicious host may alter the code of agent by this it will start other task rather than the task assigned by its creator. A host may also try to change the data contained by agent.

IV. SECURITY MECHANISMS TO PROTECT MOBILE AGENTS

A number of mechanisms that have been developed to protect the mobile agents. These mechanisms can be categorized in four types of protection:

1. Mobile agents can only migrate to the trusted nodes/hosts of the system.
2. Organizational methods may be employed to protect agents (i.e. create a close system where only trusted parties can be host.)
3. To insure the integrity of an agent we can use tamper-resistance hardware.
4. We can use cryptographic protocol to ensure the security of mobile agent.

Therefore, we will describe the generic security technologies and research efforts to counter the identified threats occurred due to mobility property of mobile agent.

There are two approaches to protect mobile agent as follows [26]:

1. Detection mechanism: To detect any unauthorized modification of an agent.
2. Prevention mechanism: Use security techniques to prevent the unauthorized access of code and data.

Table 1 shows the some of the countermeasures that have been created to protect mobile agents from malicious host and states whether the mechanism is aimed at detection or prevention.

Some of techniques are kept in prevention category employ methods such as: replication of mobile agents, the use of digital signatures to detect tampering and various cryptographic schemes. The goal of each of these mechanisms is to determine when an agent is attacked. The preventive technique has basic aim to hide data, code and flow control from hosts where they execute. On the other hand, detection techniques try to detect any attack when it will be occurred.

<table>
<thead>
<tr>
<th>Countermeasures</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial Result Encapsulation</td>
<td>Detection</td>
</tr>
<tr>
<td>Mutual Itinerary Recording</td>
<td>Detection</td>
</tr>
<tr>
<td>Execution Tracing</td>
<td>Detection</td>
</tr>
<tr>
<td>Time Limited Black Box</td>
<td>Prevention</td>
</tr>
<tr>
<td>Computing with Encrypted Functions</td>
<td>Prevention</td>
</tr>
<tr>
<td>Environmental Key Generation</td>
<td>Prevention</td>
</tr>
</tbody>
</table>

E. Partial Result Encapsulation

Partial result encapsulation is detection technique that detects any tamper, which occurs in the result of the agent at different platform or hosts. In this technique, the result of the agent at the each host is encapsulated. The encapsulated result helps in the verification either when the agent returns to the home platform or possibly at intermediate points.

Partial result encapsulation technique uses different cryptographic primitives such as encryption, message authentication code and hash function. Sliding encryption technique is used to provide confidentiality to the result. Here the mobile agent uses the public key of its originator to encrypt the result at each platform and generate the cipher text. Later, when agent returns to its home and it decrypts the cipher text by using corresponding private key there. This technique is valuable where space is important such as smartcards and key size is larger than the gathered data needed to encrypt by the mobile agent [4][5].

Yee proposed Partial Result Authentication codes (PRAC) [6] technique that is used to encapsulate the partial result with Message Authentication Code. The mobile agent and its originator help to generate list of secret keys before it moves to next host in its itinerary. In this PRAC technique, mobile agent uses the particular secret key from list of secret keys for particular host to encapsulate the results. Once encapsulation process is complete at current host, the mobile agent destroys the used key before its migration to the next host. Erasing secret keys before agent move to next host that ensures the previous results are secure. This ensures the forward integrity property, which states that, no future visited host will able to modify the previous results. Since the agent originator remains, contain the list of secret keys and verify the partial results. Therefore, it improves the integrity in mobile agent system [6].

There are some limitations in this approach. The most critical problem occurs when the malicious host exposes secret keys or key generating functions then it can tamper the results without any possibility of detection. Second, it does not secure the code of agent or aspects of mobile agent. The PRAC is oriented toward integrity and not confidentiality; the partial result can be viewed by any platform visited, although this is easily resolved by applying sliding key or other forms of encryption.

F. Mutual Itinerary Recording

In this technique, the path history of mobile agent is maintained to detect tampering attack on the mobile host by the malicious host. Itinerary is tracked by the two cooperating agents [7]. The agent conveys the information about the current platforms, last and next platforms to the cooperating peer through the secure channel. This peer keeps the records of the path and takes the important action when any unfavorable condition occurs. Main logic behind this scheme is based on the assumption that only a few agent platforms are hostile, and even if an agent comes across one, the platform is not likely to collaborate with other malicious platform being visited by the peer. Thus, the
malicious behavior of an agent platform can be detected by dividing up the operations of the application between two agents. There are some drawbacks in this technique. Firstly, it is a costly operation to maintain the authenticated communication channel. Secondly, this technique is not capable to decide which one of the two hosts may be responsible for killing agent.

G. Execution Tracing

Operating systems, scriptable applications, mobile codes all software should be capable to secure them self from malicious code. Execution tracing is a detection technique; this technique is used by all kind of software now going to use this technique to make mobile agents more secure from malicious host. Many researchers have been present many techniques which are used in execution tracing one of them is security automata. Discuss here security automata technique which is widely accepted by many researchers. But many security policies like Chinese wall policy [9, 10, 11], one out of k-authorization policy, and low water mark policy restrict the execution tracing and allow only to trace shallow history of previously grant access events, cause of this we shall also discuss shallow history automata [8] here.

1) Shallow History Tracing

This section includes the Execution tracing policies, security automata and then our final solution cause of many execution policies shallow history automata which is an extension of security automata. The class of to prove to be a proper subset of the class of Execution Tracing security policies, thereby confirming the claim that subclasses of Execution Tracing security policies can be described through the restriction on accessible information. To fix thoughts, the notion of Execution tracing enforceable policies and its categorization via security automata is discussed in C.1.1 and C.1.2 respectively. Shallow history automata are then discussed in section C.1.3.

2) Execution Tracing Security Policies

Let \( \Sigma \) is a countable infinite or finite set of access events. A policy (P) is a set \( P \subseteq \Sigma^* \) is a finite sequence of access events. An execution tracing enforceable policy is a non-empty prefix closed policy, a policy \( P \) is prefix closed policy if it satisfies the condition mentioned below:

\[ \forall u \in \Sigma^* : u \notin P \Rightarrow ( \forall v \in \Sigma^* : uv \notin P) \]

Let a prefix \( \omega \) is the set of all prefixes of \( \omega \) including \( \omega \) itself that is:

\[ \text{Prefix}(\omega) = \{ u \in \Sigma^* \mid \exists v \in \Sigma^* : uv = \omega \} \]

Now easily we can see the following equivalent characterization of prefix closed policies.

\[ \forall \omega \in \Sigma^* : \omega \notin P \Rightarrow \text{prefix}((\omega)) \subseteq P \quad (1) \]

3) Security Automata

Security automata is a variant of Buchi automata is defined [8]. A security automata is a quadruple \( (\Sigma, Q, q_0, \delta) \) where:

- \( \Sigma \) is a finite set of access events.
- \( Q \) is a finite set of states.
- \( q_0 \in Q \) is an initial state.
- \( \delta \) is an transition function maps: \( Q \times \Sigma \rightarrow Q \).

The notion of acceptance in security automata is different from regular finite automata. In regular finite automata final state is explicitly defined but in security automata (SA) final state is not explicitly defined. Security Automata (SA) accepts an access event sequence if transition is defined for every event in the sequence. The notion is mention as follows. Let us consider a given Security Automata (SA)

\[ M = (\Sigma, Q, q_0, \delta) \]

now following notations are defined for

\[ q, q' \in Q, a \in \Sigma, \omega \in \Sigma^* \]

\[ q \xrightarrow{a} M q' \text{ if } (q, a) = q' \]

\[ \Longleftarrow}_{M} q^* \text{ if } \exists \omega \in Q, q = M q^* \Lambda q^* \xrightarrow{a} M q' \]

We can say that Security Automata (SA) \( M \) accepts an access events \( \omega \) if \( q_0 \omega a q \) for some \( q \in Q \). The policy \( P \) is defined as the set of all sequences accepted by \( M \):

\[ \{ \omega \in \Sigma^* \mid \exists q \in Q : q_0 \xrightarrow{a} q \rightarrow_{M} q \} \]

We can easily see that such a set is always prefix-closed and non-empty that is policy \( P \) on security automata \( M \) denoted as \( P(M) \) contains \( e \) and satisfies equation 1. Conversely we can say for any given prefix-closed and non-empty policy \( P \) and there is a security automata \( M \) so that \( P = P(M) \). Now let us consider a security automata to see this is \( (\Sigma, \Sigma^*, e, \delta) \) where \( \delta P(\omega, a) \) is defined to be \( \omega a \) if \( \omega, \omega a \in P \) such a security automata recognize \( P \). Consequently the class of execution tracing enforceable policies coincides with class of policies recognized by the security automata. We will use the above security automata (SA) to recognize \( P \) the canonical security automata for policy \( P \), and denote it by \( SA(P) \) Intuitively, the state of a Security Automata represents the information which execution monitor tracks for execution tracing. It shows the internal data structure which is maintained by the execution monitor across the subsequent access granting decisions. The image of the transition function captures the updating procedure of the internal data structure, while domain of the transition function captures the logic of access granting decisions. Notice that the canonical Security Automata (SA) tracks the all history of previously granted access events.

4) Shallow History Automata

Let us consider the set of all finite subsets of set \( S \) is \( F(S) \) [8]. A finite subset of \( \Sigma \) is the shallow access history or simply shallow history a member of \( F(\Sigma) \). Our main task is to define automata that track only shallow history of previously granted access events. A shallow history automata (SHA) is a special kind of security automata (SA) in the form of,

\[ (\Sigma, F(\Sigma), H_0, \delta) \]
been proved that there is no SHA (Shallow History) amounts to specify its domain as a subset of $F(\sum)$. Let us consider policy (P) that no shallow history automata N is such that $\delta$ already defined at $(H,a)$. That is, the transition function ($\delta$) of shallow history automata is uniquely précised by listing all the points at which it is defined. A policy recognized by some shallow history automata (SHA) is said to be SHA enforceable. Therefore, it has been proved that there is no SHA (Shallow History Automata) is equally expressive than a security automata (SA) at any policy P.

**Theorem:** Fixing the set $\Sigma$ of all possible access events, there is a security automata M and a policy P, so that no shallow history automata N is such that $P(M) = P(N)$.

**Proof:** Let us consider $\Sigma = \{a, b, c, d\}$ and policy P = prefix (abcd) $\cup$ prefix (badc). The policy (P) is prefix closed Q and non-empty and it is recognizable by its canonical security automata. Let us consider policy (P) is recognized by shallow history automata N. Now consider $H_0$ is the initial state of N. The following transitions are valid [8]:

$$
\begin{align*}
H_0 \xrightarrow{a} M\{a\} \cup H_0 & \quad \xrightarrow{b} M\{a,b\} \cup H_0 \\
\quad \xrightarrow{c} M\{a,b,c\} \cup H_0 & \quad \xrightarrow{d} M\{a,b,c,d\} \cup H_0
\end{align*}
$$

However with above transition N also accepts abdc and bacd:

$$
\begin{align*}
H_0 \xrightarrow{a} M\{a\} \cup H_0 & \quad \xrightarrow{b} M\{a,b\} \cup H_0 \\
\quad \xrightarrow{c} M\{a,b,c\} \cup H_0 & \quad \xrightarrow{d} M\{a,b,c,d\} \cup H_0 \\
H_0 \xrightarrow{a} M\{a\} \cup H_0 & \quad \xrightarrow{b} M\{a,b\} \cup H_0 \\
\quad \xrightarrow{c} M\{a,b,c\} \cup H_0 & \quad \xrightarrow{d} M\{a,b,c,d\} \cup H_0
\end{align*}
$$

Now from the contradiction policy P is not shallow history automata (SHA) enforceable. Therefore, it has been proved that there is no SHA (Shallow History Automata) is equally expressive than a security automata (SA) at any policy P.

**H. Time Limited Black Box**

Code obfuscation technique is viable approach to secure the mobile agent code. In this technique, obfuscator transforms the code into more difficult level to understand with identical functionality. It aims to make anti-tamper mobile code which hard to understand or analyze by malicious hosts. There are many useful obfuscating transformations: Layout obfuscation, Data obfuscation, Control obfuscation, preventive obfuscation. First, Layout obfuscation may change in the code like scramble identifiers, change formatting, remove or add comments. Second, Data obfuscation may change in the storage obfuscation and encoding of data, aggregation like modify inheritance relations or split data and ordering. Third, Control obfuscation tries to reorderd statements, loops or expressions and change in computations like extend loops conditions. Last Preventive obfuscation tries to find out weaknesses in current decompiles or deobfuscators and inherent problems with the deobfuscation techniques [13] [14].

Hohl [12] proposed obfuscation technique, time limited black box. The goal of a time-limited black box is to scramble all of the information or code contained in an agent from others. The only information that can be obtained from an agent is the input to the agent and its output. It reflects in Fig. 3. The code and data contained in the agent is obfuscated so that it will take an attacker a long period for understanding the code of the agent. The aim of using obfuscated code is that a host will execute the code and have no idea what the code is actually doing. This technique protects the agent and data within the time interval.

**Fig.3 Time Limited Black Box Property**

An agent is time limited black box if:

1. For a certain fixed known time interval.
2. Data and code of the agent specification cannot be read.
3. Data and code of the agent specification cannot be tempered.
4. Attacks after the protection interval are possible, but these attacks do not have effects.

There are some limitations of black box security. There is a possibility for host to deny the execution and may return false result to mobile agent. This technique is complex and costly in the terms of execution and transmission speed.

**I. Environmental Key Generation**

Mobile agents are prepared for executing on various hosts with different environmental security conditions. The aim of this technique is improve the security of mobile agents and allow their execution on various environmental security conditions. Environmental key generation is a preventive security technique of mobile agent. Thus in this technique we
proposed an adaptive trust mechanism. It is basically based on the dynamic interaction between mobile agent and environment. By using the environmental key information collects in dynamic environment by agents from various hosts and vice-versa. This key informs to the host about trust degree and allows the mobile agent to for its execution. Trust estimation based on various parameters values.

As we already know the protection of mobile agents from malicious host is a challenging research area. Several approaches already have been already discussed like tamper proof hardware, function finding and black-box but these approaches have already some limitation and some approaches are costly. Therefore, we described here an approach which is neither limited nor more costly and also has an acceptable level of security name of this approach is Environmental key generation [15].

**Principles of this approach** Environmental key generation approach is based on a protocol (mobile agent code protection protocol) [16] and a control technique to improve trust and increase security. So here discuss the mobile agent code protection protocol and then discuss the properties of a secure environment. Now consider some assumptions:

1. Firstly the customer and service provider build a contract.
2. Some confidential information is known by the service provider which concerns the customer (e.g. contract reference).
3. The host always knows something about the environment and about itself that is not known by agent.
4. The information that is known by customer has an incidence on the agent's owner decision on
5. Execution on the requested (i.e. execute or not the requested service).
6. Mobile agent always has to be calculating the environmental key and does not consider that private information of host is correct or not.

**Mobile agent code protection protocol** The aim of this protocol is to secure code of a mobile agent against malicious hosts. The environment key does the center of our task since it is the trust degree of the target host. When the customer requires a service, it sends request to all the service providers. These providers send their acknowledgements with their proposals. Now the customer analyze all proposals and then select the best proposal and inform the respective service provider which generates public and private keys, and assigns particular key and the sufficient abstract expression to every behavior of the mobile agent [15] Then mobile agent moves to the customer host. The main steps of this protocol are mention in the Fig.4.

- Agent starts interacting in the environment in order to obtain the required information to generate the environmental key.
- With the public key of service provider customer encrypts the environment key and sends it to the provider.
- The service provider understands the received key. It identifies the key, selects the corresponding abstract expression, encrypts this expression with the environment key and sends it to the costumer.
- With the environment key customer tries to decipher the abstract expression. If it success then executes the requested service.

A secure environment An environment is secure or not for this test observation required and this observation is also use to enhance security and establish trust. If mobile agent does not establish trust, it uses observation to prevent itself from host or at least detect its misbehavior. Any host if it knows that agent is observing it then it tries to be more reliable [18]. According to Josang et al. trust is to which one party is willing to depend on somebody, or something, in a given situation with a feeling of relative security, even though negative consequences are possible [17].

Several parameters and malicious behaviors relies trust of host. We need several questions to define trust these questions are:

- For emitting the right opinion how can the agent perceive its reception environment?
- How can various observations aggregated to generate the environment key?
- How can this environment key exactly define the category of customers?
- How can this key inform about origin of failure in case of misbehavior?

**Key generation** Now we describe here cryptographic methods used to generate the environment key. Let us consider a set $E$ of $n$ abstract expressions as $E = \{E_1, E_2, E_3, E_4, ... E_n\}$ these expressions used to implement the different behaviors of mobile agent and let a set $A = \{A_1, A_2, A_3, A_4, ... A_p\}$ is the set of $p$ adaptable modules (include dummy) that is include in different implementations[5]. Each $E_i (i \leq n)$ is a sequence call of $A$’s subsets and also can be viewed as sequence of bits (each bit indicates the specific modules). If there are $P$ modules then total possible
combinations is $2^p-2$. Each combination is associated with an abstract expression (without concluding the empty expression). The environment key $K_j$ is generated for each expression $E_j$. The key definition is used to collect information at the level of host with the identifier of mobile agent which is unique. It is based on hash function and public key cryptography. Let us consider the couples of public and secret keys.

Host keys (Ph, Sh).

Agent owner keys ($P_0, S_0$).

As soon as the mobile agent riches at the customer host, it executes some actions which are using in trust acquisition explain in algorithm 1 given below [15].

### J. Algorithm 1 The Mobile Agent Behavior

1. Gather data correspond to parameters values let us consider \{d_1, d_2, d_3, d_4, ..., d_k\} is the set of collected data.
2. Apply secure hash one way function (SHS) to each data. For (i = 1 to k) do $M_i = H(d_i)$ end for.
3. Concentrate on all digits and find $M = \{M_1, M_2, M_3, ..., M_k\}$.
4. Encrypt $M$ with agent’s owner public key $P_0 (M)$.
5. Send the signed message $SM=S_0 (P_0 (M))$ to service provider.
6. In the 3rd step result apply hashing. Let us consider $D = H(M)$ is the final digest.
7. Apply $D \oplus id$ (here id is the unique identifier of mobile agent) to generate the environment key $k_j$ which will be used to decrypt the $E_j$ (abstract expression).
8. Receive an abstract expression.
9. Now decrypt the received abstract expression with the help of $k_j$.
10. If decryption done successfully then executes the selected services.

Service provider also executes some actions in order to obtain customer trust degree, explain in algorithm 2.

### K. Algorithm 2 The Service Provider Action

1. Received signed message (SM) = $S_0 (P_0 (M))$.
2. Calculate $P_0 (M) = P_0 (S_0 (P_0 (M)))$.
3. Calculate $M = S_0 (P_0 (M))$.
4. Obtain k digest $H(d_1), H(d_2), H(d_3), ..., H(d_k)$.
5. Check the obtained digest with the digests present in database.
6. By calculating the value of $T$ estimate trust worthiness [8].
7. Compare the trust values with the interval values and select the actions to be undertaken.
8. According to selected service select abstract expression; Let $E_j$ is the selected abstract expression.
9. Apply hashing to the result of 3rd step let $D = H(H(d_1), H(d_2), ..., H(d_k))$ be the final digest.
10. Apply $D \oplus id$ (where id is the unique identifier of mobile agent) to generate key $k_j$.
11. With the help of $k_j$ encrypt the selected abstract expression $E_j$.
12. Sign on encrypted abstract expression and send out to the customer.

We calculating environment key customer side as well as service provider side in order to protect it and avoid transmitting it.

### L. Computing with Encrypted Function

This is a preventive technique to secure mobile agent from malicious host. The main goal of this technique is to determine a method by this code of a mobile agent can safely compute cryptographic primitives. This approach is based on three basic techniques:

1. Homomorphism encryption scheme (HES).
2. Three address code.
3. Function composition (FnC).

#### F.1 Three address code

Today’s many high level languages like C, Java contains compiler to convert the source code into target code [19]. Compilers use several phases like lexical analysis, syntax analysis and semantic analysis etc. Most of the compilers generate intermediate code before generating the target code. Three address code is also one form of intermediate representation. Let us take an example the source code contains expression like $a + b * c$ then it may be translated in the following sequence:

$p1 = b * c$;
$p2 = a + p1$;

Where $p1$ and $p2$ are compilers generated temporary variables. Three address code contains three addresses two for operands and one for operators.

#### F.2 Homomorphic encryption scheme

Many researchers find the limitations of encryption system that is when data is decrypted then it is no more secure. Thus researchers develop a new technique of cryptography where authorize user is enable to compute encrypted data without decryption which is called privacy homomorphism [20]. After this, Sander and Tschudin describes additive-multiplicative homomorphism, which is a type of privacy homomorphism [21] [22]. Additive-Multiplicative homomorphism technique ensures that the computation result of two unencrypted values is same as the computation result of two encrypted values. Now we are going to describe the properties of additive-multiplicative homomorphism on the basis of Sander and Tschudin work. Let us consider there are two rings $G, H$ and an encryption function $E$ as $E: G -> H$.

- **Additive Homomorphic** If there is an efficient algorithm PLUS to compute $E(x+y)$ from $E(x)$ and $E(y)$ that does not reveal $x$ and $y$.
- **Multiplicative Homomorphic** If there is an efficient algorithm MULT to compute $E(xy)$ from $E(x)$ and $E(y)$ that does not reveal $x$ and $y$. 
• Mixed Multiplicative Homomorphic

If there is an efficient algorithm MIXED-MULT to compute $E(x*y)$ from $E(x)$ and $y$ that does not reveal $x$.

Homomorphic schemes described above allow only two types of operations that is addition and multiplication. One important thing is there that is one to many relationship is there that is there may be generate multiple cipher text messages $E(x)$ from a single plaintext message $x$ i.e.

It is possible for two cipher text messages $E_1(x) \neq E_2(x)$ but plain text after decryption always $D(E_1(x)) = D(E_2(x))$. Another important point is that only few elements (only one element is desirable) that satisfies the last Mixed Multiplicative homomorphic property otherwise last and second property will produce an anomaly $y = E(y)$. Thus only one integer that is $1$ (multiplicative identity) should satisfy the last property $E(x*y) = E(x)*y$.

F.3 Function Composition

Sander and Tschudin argue that computing with encrypted functions cannot be accomplished by only additive-multiplicative homomorphism, but also by mathematical analogues like composite functions [23].

Lets consider a scenario: Alice wants to evaluate a linear map $A$ by using Bob’s input $x$ in Bob’s computer. Alice does not want to show linear map $A$ to Bob, so Alice picks at random matrix $S$ (invertible), and then computes $B := SA$ and sends $B$ to Bob. After receiving $B$, Bob computes $y = Bx$ and sends $y$ return to Alice. After receiving $y$, Alice computes $S^{-1}y$ and get the result $Ax$ without having revealed $A$ to Bob.

Here define the $f(x)$ as a composite function is represented by $f(x) = g \circ h$ or $f(x) = g(h(x))$, it is derived by using the output of a function, $h(x)$, and apply as the input to another function, $g(x)$.

In the Fig. 5, Alice is the agent owner and has an $h(x)$ function, that Alice wants to evaluate in Bob’s computer with Bob’s input $x$, but Alice does not want to show anything about function. Alice chooses $g(x)$ an invertible function, and creates $f(x)$ a composite function and sends it to Bob. Bob does all the computation with input $x$, and sends the result return to Alice. Bob is not capable to determine the function $h(x)$ (i.e. Owner's function), because what Bob can see this is only the $f(x)$ (composite function). Only Alice can extract the exact result of $h(x)$ from the result of $f(x)$ by using $f(x)$ into the inverse function of $g(x)$ (that is $h(x) = g^{-1}(f(x))$).

V. COMPARATIVE STUDY OF DETECTION TECHNIQUES

Only three detection techniques for securing the mobile agents these techniques are: Partial result encapsulation, Mutual itinerary recording and Execution tracing. Every technique has some limitations and no technique is there to fulfill all objectives. Table 2 summarizes the study of these three mechanisms.

Comparative study of preventive techniques of three preventive techniques are: Environmental key generation, Time-limited black box and Computing with encrypted functions. Table 3 summarizes the comparative study of preventive technique.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Prevent Unauthorized Access of Information</th>
<th>Prevent Masquerading</th>
<th>Prevent Denial of Service</th>
<th>Prevent Emasquerading</th>
<th>Prevent Copy and Reply</th>
<th>Detect Tampering</th>
<th>Implementation Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutual Itinerary Recording</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Partial Result Encapsulation</td>
<td>PARTIAL</td>
<td>YES (digital signatures)</td>
<td>YES</td>
<td>YES (digital encryption)</td>
<td>YES (digital signatures)</td>
<td>PARTIAL</td>
<td>YES</td>
</tr>
<tr>
<td>Execution tracing</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES (time limited)</td>
<td>PARTIAL</td>
</tr>
</tbody>
</table>

TABLE 2

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Prevent Unauthorized Access of Information</th>
<th>Prevent Masquerading</th>
<th>Prevent Denial of Service</th>
<th>Prevent Emasquerading</th>
<th>Prevent Copy and Reply</th>
<th>Detect Tampering</th>
<th>Implementation Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environmental key generation</td>
<td>PARTIAL</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>PARTIAL</td>
<td>NO</td>
</tr>
<tr>
<td>Time limited black box</td>
<td>YES (time limited)</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES (time limited)</td>
<td>YES</td>
</tr>
<tr>
<td>Encrypted functions</td>
<td>PARTIAL</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>PARTIAL</td>
<td>NO</td>
</tr>
</tbody>
</table>

TABLE 3
Partial Result Encapsulation PRAC is oriented towards integrity and not confidentiality, the collected set of partial results can be observed by any platform visited.

Executing Tracking A drawback of this approach consists in the size of all the created logs by the hosts (each host may execute a large number of mobile agents). Another drawback consists in the difficulty of the logs management. The detection process is only triggered occasionally, based on suspicious results or other factors. Also, the size of the logs could get unmanageable.

Mutual Itinerary Recording This is a technique to observe the path history of mobile agent in relation how it visits the platform. This technique is very limited in practical realization due to problems of expectation that the programmer to know all the nodes that the agent will visit. It is a costly operation to maintain the authenticated communication channel.

Time Limited Black Box Security This approach tries to generate a ‘black-box’ out of agent code by using code obfuscating technique. Code obfuscation technique used to provide code confidentiality. Code obfuscation has capability to scramble the agent’s program and make it difficult to manipulate and understand.

Since an attacker needs time to examine the black-box code before it can attack the code, the agent is protected for a certain interval. After the ‘expiration interval’, the agent and the data it transports become invalid. If the agent is successfully converted into a black box, then the hosts cannot in any direct way interfere with its execution. Thus, a whole lot of threats such as eavesdropping, alteration of state etc. can be solved. But there is a possibility for host to reject the execution and may return false result to agent. This technique is complex and costly in the terms of execution and transmission speed.

Computing With Encrypted Function This approach enhance the security by avoiding the decryption of encrypted data to generate a new encrypted value from the new data and previously encrypted value. So the original data can be transmit, in encrypted form, to hosts that can perform the obligatory computation, while conserving the privacy of not only the encrypted data, as in privacy homomorphism, but the privacy of the keys as well. But finding suitable encryption schemes that can transform arbitrary functions is a challenge. This scheme doesn’t prevent denial of service, replay, and experimental extraction.

Environmental Key Generation The environmental key generation can protect the code and data from integrity and privacy attacks, but this approach also has weaknesses. First, this approach is vulnerable to group conspiracy attack. Second, data channel protection is another security issue. Third, although this approach can improve the integrity and the privacy for its code and data, it does not provide any protection for results. Fourth, once the code and data are decrypted, they can be attacked by a malicious host who can insert his or her own decrypting routine and data channel for new hosts. Clueless agents are proposed as solution to prevent code and data disclosure [8].

### Table 4

<table>
<thead>
<tr>
<th>Proposed Mechanism</th>
<th>Time limited black box</th>
<th>Computing with Encrypted functions</th>
<th>Environmental Key generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method used for hiding code</td>
<td>Obfuscation</td>
<td>Homomorphic Encryption</td>
<td>Keys are generated from one-way hash functions</td>
</tr>
<tr>
<td>Limited by time interval</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Code is executable without decrypting</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Entire program hidden</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Technique is mathematically provable</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

### REFERENCES


