Design and Implementation of Inter-operable and Secure Agent Migration Protocol

Shakir Ullah Shah¹, Jamil Ahmad², Najeeb-ur-Rehman³
¹Department of Computer Science, Iqra University Islamabad, Pakistan
²Department of Computer Science, National University of Computer & Emerging Sciences (NUCES), Peshawar, Pakistan
³Senior Member IEEE, Department of Computer Science, University of Science and Technology (KUST), Kohat, Pakistan

Abstract: Mobile agent technology is an active research topic and has found its uses in various diverse areas ranging from simple personal assistance to complex distributed big data systems. Its usage permits offline and autonomous execution as compared to classical distributed systems. The free roaming nature of agents makes it prone to several security threats during its transit state, with an added overhead in its interoperability among different types of platforms. To address these problems, both software and hardware based approaches have been proposed to ensure protection at various transit points. However, these approaches do not ensure interoperability and protection to agents during transmission over a channel, simultaneously. In this regard, an agent requires a trustworthy, interoperable, and adaptive protocol for secure migration. In this paper, to answer these research issues, we first analyse security flaws in existing agent protection frameworks. Second, we implemented a novel migration architecture which is: (i) fully inter-operable compliance to the foundation for intelligent physical agents (FIPA) and (ii) trustworthy based on Computing Trusted Platform Module (TPM). The proposed approach is validated by testing on software TPM of IBM, JSR321, and jTPMTools as TPM and Trusted Computing Software Stack (TSS) interfaces. JADE-agent framework and JADE Inter-Platform Mobility Service (JIPMS). Validation is also performed on systems bearing physical TPM-chips. Moreover, some packages of JIPMS are also modified by embedding our proposed approach into their functions. Our performance results show that our approach merely adds an execution overhead during the binding and unbinding phases.

Keywords: Information Security, Multi-Agent Systems, Inter-Platform Agent Mobility, JADE, Trusted Computing.

Received October 5, 2018; accepted February 24, 2019

1. Introduction

Mobile agent oriented technology is quite popularly used in e-commerce [35], search engines [36] [46], web-crawlers [33], scheduling problems [22] [30], parallel computing [24], health monitoring [39], network administration [34] and many other areas [41]. The mobility nature of the agent makes it more valuable while moving from one platform to another to perform different tasks on behalf of its owner. Along the course of migration on open networks, an agent may have private and confidential data that can be shared with only intended hosts or other agents. Maintaining trust in open networks is an open challenge and it is not a viable solution to provide a secure channel to each and every agent. Other problem with the agent is that when it visits different hosts, then a host may alter its secret data or even whole code. In such case an agent will not remain trustworthy. So at such scales, security and interoperability are the major challenges that affect usability and reliability [37] of Multi-Agent Systems (MAS).

Three levels of security are required for MAS; host, channel, and agent security. In an open channel, a mobile agent is prone to Active [4], man in the middle (MITM) [27], and phishing [25] attacks which decreases level of trust. In this work, trusted and interoperable agent migration protocol based Trusted Computing is proposed. The Trusted Computing Group (TCG) prescribes the standards to increase level of trust based on a tamper resistant cryptographic hardware known as a Trusted Platform Module (TPM) [42]. For a proof of concept, Java Agent Development (JADE); a popular multi-agent Foundation for Intelligent Physical Agents (FIPA) compliant development framework [13] is used with TPM. Trust models of FIPA and their brief general security mechanisms for agent based systems is provided in [32]. The intrinsic TPM key hierarchy strengthens the security of the framework such that no device other than the intended host and platform can unbind or decrypt a private part of a binding key. It works by storing the Storage Root Key (SRK) in a non-volatile memory inside the TPM so that the whole tree
structure becomes secure. As JADE does not support Inter Platform Mobility, JIPMS [10] is used for this purpose.

In the next section (Section 2), required background details pertaining to FIPA, JADE, and TC is provided. The section covers a literature review to demonstrate the progress made so far. Section 3 provides proposed solution and section 4 provides the implementation details of the proposed protocol. This is followed by results and discussion section 5. Section 6 provides conclusion and future directions.

2. Background and Literature Review

A number of security attacks in the literature about the domain of mobile agent frameworks have been discussed in [36] [38] [20] [44]. The simplest form is an agent to agent attack involving one malicious agent attacking another on any host (local or remote) [6]. A malicious agent can also initiate an attack on its hosting platform in order to gain privileged access to the host. In this case, the attack is directed from malicious agent to an agent platform. But the counterpart can also be true. As an example, malicious platform can terminate the migration process an agent and also truncate the data gathered by it in the form of a Colluded Truncation Attack [23] [26]. Another variation involves attacks directed from one agent platform to another one. Other categories involve eavesdropping, denial of service, masquerading, unauthorized access, copy and replay, alteration, and man-in-the-middle attacks [36].

From time to time [21] [6] [23], general solutions for integrity of agents and platforms are provided. TCG [49] provides solution for the agent security by introducing freezing mode. The secure session key is managed in a secure session database. For this purpose, a new layer of Transport Guard Process (TGP) is merged with the protocol and it is responsible for an agent's migration along with an Agent Control Center (ACC). The agent freezes itself at source host, half freezes itself over the network and unfreezes itself at the destination host. At the time of freezing, credentials such as timestamp, sequence number (nonce for protecting from replay attack), information/message digest (SHA-1), and digital signatures for integrity of digest are stored in the database. Authors in [14] developed a FIPA compliant trial involving a secure and efficient code distribution service known as JADE-S (JADE Security Module) [7]. The model uses Java Authentication and Authorization Service (JAAS), Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE). User's credentials are store in plain form in a text file. Some other deficiencies in this model are discussed in [45].

Some software based approaches are also proposed but they are not able to provide fool proof security [9]. For this reason, TCG has proposed hardware solutions in form of TPM. Examples of software based solutions is a JADE and TPM4Java [19] which based on Secure Migration Library for Agents (SecMiLiA) [28] that bears functionality to address the malicious host problems. A downside of the library is that it works with a single key only at a time and as a result, it is not scalable. Code integrity is preserved using an approach used in [48]. In this approach, before migration, the integrity of the code is measured by taking the hash using SHA-1. At receiving side again hash is taken again and recorded. The code is allowed to resume for execution only after insuring the agent's integrity. [47] proposed different approach for code integrity. In this approach the whole structure is encrypted using public part of an Attestation Identity Key (AIK) of a destination host. A procedure known as Sealing and Unsealing is provided by TPM and used and discussed in [29]. The procedure is usable on local machines only and is strict in nature as it is locality dependent. A similar approach that guarantees integrity and confidentiality is also proposed in [57] [11] where hosts are authenticated using Public Key Infrastructures (PKI). A mobile agent-based crawler that promises to protect agents from malicious hosts is proposed in [43]. A self-reliant mobile code proposed in [40] promised to protect the agents selfishly. Symmetric keys are communicated in a secure way with digital signature. AI based approaches for protecting agents has been discussed in [8]. A robust and fault tolerant approach has been proposed in [31] that is based on security of code in a sedentary agents.

Interoperability in mobile agents can be handled at different levels such as executable code, platforms, and communication protocols [3]. The most common approach for dealing with interoperability is carried out at the application level. Our research focuses on interoperability issues arising due to migration, and for that the focus is on the communication protocols. A specification for FIPA compliant communication protocols utilizing Agent Communication Language (ACL) is discussed in [16] [17] [2]. For mobility, the Simple Mobility Protocol and Full Mobility Protocol stacks are used and described in [18].

The migration process as per FIPA standards requires three steps. The first step is the negotiation phase, where source and remote platforms communicate via ACL messages using the FIPA’s Request Interaction Protocol (RIP). In the second step, configuration phase, the source and remote platforms share configurations and parameters. Lastly, in the execution phase, the selected transfer protocol is used for migration [12]. To achieve interoperability, mobility is placed under the Pre-Transfer, Transfer, and Post-Transfer protocols [13] [15] illustrated. In our implemented protocol, the same steps are used for mobility as described in [3]. Protocols which are used for transferring agent from one host to another are the Push Transfer, Pull Transfer, Push Cache Transfer, on-Demand Transfer, and Fragment Transfer.
Protocols [12]. The proposed protocols discussed so far pertain to homogeneous platforms. For heterogeneous hosts, a detailed is in [11].

3. Proposed Solution
A new secure, flexible, and fast protocol for agent migration between a source and destination is proposed in this section using the Trusted Computing. The protocol uses symmetric keys, nonce, and timestamps to keep off replay attacks. Though the key management is out of the scope of this work but only an abstract idea is given for the sack of understanding. The Binding key Certification Authority (BKCA) is responsible for the management of RSA Binding Key certificates. The migration model of the process is designed as a layered architecture and is illustrated in Figure 1.

The migration model of the process is designed as a layered architecture and is illustrated in Figure 2. Agent Management System (AMS) communicate with a remote platform for the migration of an agent. It performs all requests and proceeds to the next mobility steps. FIPA Request Interaction Protocol (RIP) is the default migration protocol. All requests of an agent pertaining to migration are served. Similarly, it is responsible for serving all requests communicated by the Trusted Agent Migration Manager (TAMM).

The proposed agent, TAMM, acts like an interface and provides security for trusted T-move. The binding process involves TPM keys creation, return information about TPM status, return TCG compliant unique ID labelled by manufacture, configure block size for both symmetric and asymmetric cryptography. At SH, when it receives public part of binding key, then it encrypts the agent and forwards it to AMS. AMS at the destination sends bound agent to TAMM to unbind.

![Figure 1. Layered Architecture of Trusted Migration Protocol.](image)

Agent is at the top of the trusted migration protocol which is an entity that wants to move from SH to DH. For migration, an agent has two options; First being requesting the AMS to process the case via JIPMS mobility procedure, and the second being making the request to TAMM to process the case under TC specifications and FIPA standards. The former is termed as simple move and allows greater degree of flexibility, while the latter is termed as T-move and allows greater degree of security.

The Agent Middleware provides support and functional environment just like a standard platform for any real implementation of a multi-agent system. This is done by having support for the agent’s entire life-cycle along with all expected transition states. The Trusted Computing Software Stack Layout is composed of TCG specified components for the functional environment. These components include the Trusted Platform Module (TPM), TPM Drivers, TPM Device Driver Library (TDDL), TSS Core Service (TCS), TSS Service Provider (TSP), and RPC/SOAP Communications. The TPM Drivers are provided by TPM manufacturers and support all required operations of the TPM. These drivers are loaded into the kernel as modules. The TDDL provides a standard interface to communicate with TPM and also provides support for switching between user and kernel mode. The TCS lies in user space and provides support for sophisticated tasks like key management efficiently. The TSP is the top most module in TSS layered architecture and provides an attractive object oriented interface to the applications. These applications then use an interface termed as the TSS Service Provider Interface (TSPI). The TSP executes in user-space while the TCS lies in kernel-space. For inter-communication, RPC/SOAP is used, which provides services like marshalling and unmarshalling. The TPM’s protocol does not rely on security properties of the data transfer and assumes that all parts of the channel as un-trusted.

The proposed protocol follows a multi-protocol based setup which consists of Pre-Transfer, Transfer, and Post-Transfer protocol. The Pre-Transfer protocol comprises of a negotiation and configuration phase. The Transfer protocol comprises of the binding and transfer phase. The Post-Transfer protocol consists of resumption phase meant for resuming execution of an agent. Each of these phases are discussed in the coming sections.

When a AMS receives trusted migration request from a Mobile Agent (MA), see figure 2, from its Source Host (SH) to a Destination Host (DH). Then steps given below are performed: (1) SH will forward migration request to DH. (2) Let DH is agree and responds to initiator. (3). SH bind the agent before t-move which consists further steps. (a) First of It search the public key of DH in its local keys-repository, if fails then (b) It request to Binding Key Certification Authority (BKCA) for it. (c) BKCA tries to fetch it in its local repository. If success it returns, if fails. (d) BKCS request to DH and return it SH. (4) DH binds the agent using received key. (5) DH unbind the agent using its own private part of binding key. DH informs SH about the successful power-up of agent.
The proposed protocol follows a layered architecture. The architecture is further divided into different phases. These phases consist of Pre-Transfer, Transfer, and Post-Transfer protocol. The Pre-Transfer protocol comprises of a negotiation and configuration phase. The Transfer protocol comprises of the binding and transfer phase. The Post-Transfer protocol consists of resumption phase meant for resuming execution of an agent. Each of these phases are discussed in the coming sections.

3.1. Negotiation Phase

Negotiation is the first phase of the proposed model. This phase starts with a request of T-Move from an agent to the TAMM at SH (TAMM$_S$). The TAMM$_S$ forwards the move request to AMS$_S$ which communicates with the destination AMS$_D$ to get a decision for migration. Upon receipt of confirmation from AMS$_S$, TAMM$_S$ asks about the physical presence of TPM via AMS$_D$. AMS$_D$ submit the query to TAMM$_D$ via TSP. As an inquirer, TSP finds whether TPM is enabled and activated, and replies with the status of this query. The TAMM$_D$ responds to its AMS about the TPM state returned via TSP. This information is also delegated back to the SH. Both SH and DH try to agree upon a binding protocol for trusted move. Upon agreement, they select the required transfer protocol for the execution phase. Types of transfer protocols that can be selected are the Push, Pull, Push Cache, On-Demand, and Fragment Transfer Protocols. The entire process is open so that they can agree upon a common protocol with mutual understanding.

3.2. Configuration Phase

Configuration Phase the next phase of the proposed mode. The configuration phase receives all data acquired by SH and DH in the negotiation phase which includes TPM related information, binding key of DH. In T-Move the AMS$_S$ will make a request AMS$_D$ to get proof and status of DH TPM. AMS$_D$ forwards the query to TAMM$_D$, then to TSP. TSP responds the proof and status of TPM to TAMM$_D$. The TAMM$_D$ forwards the response to AMS$_D$ which in turn inform the TAMM$_S$. Now TAMM$_S$ is able to start T-Move. The TAMM$_S$ will try to locate the binding key at the local host. In case of failure the, the TAMM$_S$ will contact the BKCA. The BKCA will share only the public part of binding key.

3.3. Binding Phase

The core phase of this work is the binding phase which migrates the agent using a trusted T-Move by a TAMM. After getting the public of the binding key of DH, TAMM$_S$ binds the agent with DH according to TCG specification. Two different approaches were followed for the better results. In one approach the whole agent was bind. In another approach the agent was encrypted with AES and the key to decrypt was bind with DH. In either approach, the encrypted bounding agent is ready for T-Move. The detail of each approach is given in section 4.

3.4. Transfer Phase

This the last phase of the proposed model. In the transfer phase, agent will move to the DH using desired transfer protocol selected and agreed upon in the navigation and configuration phase by both parties i.e. SH and DH. As a result of the binding phase, the agent is already bind with DH. AMS$_S$ will forward the bind agent to AMS$_D$ which will make a request to TSP via TAMM$_D$ for unbinding. TSP will request TPM for this process. The TPM will unbind the agent using corresponding private part of the binding key along with UUID. The TSP then sends the request back to TAMM who in turn again forwards it back to the AMS as an unbound agent. The AMS then creates an agent thread and launches all required configurations. Finally, the destination AMS sends an acknowledgement to the source AMS containing status of the successful unbinding.

3.5. Execution Resumption Phase

In Execution Resumption Phase, the unbind agent will resume its execution based on its previous code, data and state. No changes were required for this phase.

4. Implementation

This section provides fine grained details for the implementation of proposed solution. In order to deploy the proposed solution, the testbed environment is created in two different ways. In first approach implementation is done on a Dell Latitude E6410 bearing a Broadcom TPM. In another approach, to confirm heterogeneity, the implementation has been tested on virtual TPM developed by IBM on Ubuntu 16.04. On software and hardware TPM enabled platforms, jTSS, jTpmTools and JADE 4.2 were configured. As JADE does not support inter-platform
mobility, therefore JIMPS was also installed on both platforms.

Let E and D shows encryption and decryption algorithm respectively. PRSH shows private part of the binding of source host while PUDH shows public part of the binding key of the destination host.

In the first approach the data of the agent is encrypted with the PRSH and then again encrypted with PRDH to secure the data and avoid repudiation as shown in Equation (1).

\[
E \left( E \left( Data; PRSH \right); PU DH \right)
\]

The public part of the binding key is extracted as a Key Binary Large Object (key-BLOB) using code outlined in algorithm 1.

**Algorithm Extract Public part of Binding-Key**

```
#Require: UUIDRand, pubKeyFileName
#Ensure: Binding Key
contxt = TPMContxt.Instance()
contxt.connect(null)
kmmg = contxt.getKeyManager()
srk = kmmg.LoadStorageRootKey(Secret.WELL KNOWN SECRET)
bdn = kmmg.createBindingKey(srk,Sec.WELL KNOWN SEC, Sec.WELL KNOWNSEC, f, f, t, 2048, null)
kmmg.storeTPMKey(srk,bdn,UUIDRand)
bdnpub1 = bdn.getPublicKey()
bdnpub = bdnpub1.Encoded()
writeFile(pubKeyFileName, bdnpub)
```

When SH gets PRDH, then SH binds the agent for T-move, code required to bind the data is listed in algorithm 2. When DH receives the bind agent then it unbind the agent. The code of unbinding is listed in algorithm 3.

In the second approach, the data of the agent is encrypted using AES encryption. The step by step procedure is shown in algorithm 4.

**Algorithm 2 Agent Binding**

```
#Require: byteData and pubKeyFileName
#Ensure: bData
rbndr = RemoteBinderImpl()
pkbbyt = readFile(pubKeyFileName)
kF = KeyFactory.getinstance(“RSA”) publicKeySpec = X509EncodedKeySpec(pkbbyt)
pub = RSAParameterSpec.getPublicKey(publicKeySpec)
inBlockSize = 209
outBlockSize = 256
DataIn = null
DataOut = null
numBlocks = (byteData.length / inBlockSize)
if byteData.length % inBlockSize != 0
    numBlocks++

if numBlocks != 0
    numBlocks = 1
if byteData == numBlocks * outBlockSize
    Arrays.fill(boundData, (byte) 0)
```

\[
E \left( p, sk \right) = c_1
\]

\[
E \left( E \left( sk, PRSH \right); PUDH \right) = c_2
\]

\[
c_1 \| c_2 = c
\]

The secret key is then decrypted via the private part of the RSA key as depicted in equation 2.

**Algorithm 3 Agent Unbinding**

```
#Require: bData, UUIDRand
#Ensure: unbData
unbData = null
contxt = TPMContxt.getInstance()
contxt.connect(null)
kmmg = context.getKeyManager()
srk = kmmg.loadStorageRootKey(Secret.WELL KNOWN SECRET)
bdn2 = kmmg.loadTPMKey(srk, UUIDRand,Secret.WELL KNOWN SECRET)
bndr = BinderImpl(contxt)
inBlockSize = 256 /*for unbinding*/
outBlockSize = 209 /*For unbinding*/
DataIn = null
DataOut = null
numBlocks = Math.ceil(bData.length / inBlockSize)
for (i = 0; (byteData.length / inBlockSize) / i < numBlocks)
    DataIn = inBlockSize
    DataIn = DataIn.fill((byte) 0)
    DataOut = outBlockSize
    DataOut = DataOut.fill((byte) 0)
    DataOut = bndr.unbind(DataIn, bnd2)
    if (i == numBlocks - 1 then
        DataOut = DataOut.unbData, i * outBlockSize, DataOut.length
```
Table 1 gives us the information of agent size in Bytes and Kilobytes (KBs). Also, it gives us the binding time and unbinding time with respect to the given agent.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Size (KB)</th>
<th>Binding Time (MS)</th>
<th>Unbinding Time (MS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25</td>
<td>6.009</td>
<td>636.494</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>9.444</td>
<td>1085.715</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>13.416</td>
<td>1714.031</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>15.574</td>
<td>2091.548</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>25.743</td>
<td>5641.002</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>36.544</td>
<td>1170.354</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>79.676</td>
<td>2163.248</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>130.633</td>
<td>4244.617</td>
</tr>
<tr>
<td>9</td>
<td>64</td>
<td>245.002</td>
<td>8461.390</td>
</tr>
<tr>
<td>10</td>
<td>128</td>
<td>439.058</td>
<td>16917.530</td>
</tr>
<tr>
<td>11</td>
<td>256</td>
<td>862.117</td>
<td>357308.620</td>
</tr>
<tr>
<td>12</td>
<td>512</td>
<td>1677.559</td>
<td>679850.489</td>
</tr>
<tr>
<td>13</td>
<td>1024</td>
<td>3740.121</td>
<td>1149530.500</td>
</tr>
<tr>
<td>14</td>
<td>2048</td>
<td>7480.237</td>
<td>2297739.000</td>
</tr>
<tr>
<td>15</td>
<td>4096</td>
<td>14561.789</td>
<td>5599927.500</td>
</tr>
</tbody>
</table>

To visualize the correlation between binding time and unbinding time with respect to the size of agent we use a line chart. Which could typically bring ease in helping us to find patterns from the given results. X-Axis: On X-axis we have the size of an agent in Kilobytes (KBs). Y-Axis: On Y-axis we have time dimension which is given in Milliseconds. Figure 3 has a line chart that shows the relation of unbinding time in milliseconds with respect to the agent size. The dotted line shows the linear unbinding time of agent with different sizes. By looking at figure 3 we have come to a conclusion that it is not feasible to deploy this solution as the execution time of binding and unbinding increases exponentially with the increase of the agent size. Since encrypting and decrypting the whole agent during binding and unbinding gets computationally expensive with the increase in the size of the agent, therefore, the solution does not remain feasible anymore. So, to overcome this problem we incorporate Advanced Encryption Standard (AES) which is a symmetric key cryptographic technique. Incorporation of AES would work in a way that at the binding time of agent we would bind the public key of the source and attach the public key with an agent. Once the agent reaches the destination the agent is decrypted by combining the public and private together. So during the whole pipeline, we are encrypting the public keys instead of agents which would save a lot of overhead. Secondly, the perks of public key encryption are that TPM ensures that the agent would only decrypt on the destination platform which ensures the integrity of the information delivered. This process typically increases the decryption performance which reduces the time latency gap in binding and unbinding execution costs. To assess the improvements by incorporating AES to our proposed method we have a look at table 2 numbers.
Table 2 shows the recorded results of binding and unbinding with support for AES. Considering the same cases, the binding takes 4 MS instead of 15 MS when the agent size is 2 Kb. Likewise, it takes only 8 MS instead of 862 MS when agent size is 256 Kb, and so on. Finally, if agent size is 4096 Kb, it is bound 117 MS instead of 14561 MS. Similarly, unbinding takes 565 MS instead of 3091 MS when the agent size is 2 Kb, 575 MS instead of 5 minutes when agent size is 256 Kb and so on until an improvement of 669 MS is noticed instead of 1.5 hours for an agent size of 4096 Kb.

Table 2 Time complexity using AES

<table>
<thead>
<tr>
<th>S.No</th>
<th>Size (KB)</th>
<th>Binding Time (MS)</th>
<th>Unbinding Time (MS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.5</td>
<td>3.926</td>
<td>595.912</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>4.157</td>
<td>590.353</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4.739</td>
<td>563.948</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4.864</td>
<td>565.545</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6.939</td>
<td>570.306</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>10.063</td>
<td>570.313</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>5.038</td>
<td>581.930</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>7.437</td>
<td>581.754</td>
</tr>
<tr>
<td>9</td>
<td>64</td>
<td>5.460</td>
<td>584.201</td>
</tr>
<tr>
<td>10</td>
<td>128</td>
<td>5.440</td>
<td>598.874</td>
</tr>
<tr>
<td>11</td>
<td>256</td>
<td>8.957</td>
<td>575.637</td>
</tr>
<tr>
<td>12</td>
<td>512</td>
<td>14.130</td>
<td>585.334</td>
</tr>
<tr>
<td>13</td>
<td>1024</td>
<td>25.859</td>
<td>601.556</td>
</tr>
<tr>
<td>14</td>
<td>2048</td>
<td>55.936</td>
<td>624.109</td>
</tr>
<tr>
<td>15</td>
<td>4096</td>
<td>117.981</td>
<td>669.442</td>
</tr>
</tbody>
</table>

Figure 4 shows efficient computational results of the proposed model of T-move of binding with AES. There are two lines, blue one shows the behavior of binding with AES Encryption and red one shows the result of unbinding using second approach.

6. Conclusion

The growing nature of the structured and unstructured date over Internet makes it very difficult to collect one’s own required data. Mobile agents are especially useful to perform functions automatically including information extraction and dissemination. On the other hand, this expanded power and usage, is also causing security concerns. The goal of this research work is to give protection to mobile agents without any sort of compromises on the integrity of information. Also, the research work enables us to conduct secure transfer and exchange of information through mobile agents. The proposed secure model mainly consists of architectural flow, migration model and sequential execution for the exchange of information with ease. For trustworthy transfer, proposed novel trusted model is used to ensure secure migration of agents from one host to another. In order to ensure the effectiveness and efficiency in actual environment, Trusted Platform Module has been used. Computing Group (TCG) which is highly recommended by experts of the domain. Hieratical security of Trusted Platform Module (TPM) by TCG is used to enhance the security and to ensure that the binding keys are not made public other than the destination host. The full migration protocol developed by us has elastic architectures which enable mobile agents to migrate with ease. The proposed method provides the user with the flexibility of choosing the security protocol instead of imposing it on the user. Our methods also let the user choose between normal and trusted migration depending upon the user requirements. Securing the communication medium during the migration of agent is beyond the scope of this research. Therefore if the keys are extracted through the medium we cannot do anything about it currently. Also, our research does not share the public part of the binding key as it is beyond the scope of our work. Our methods provide a protected and reliable environment for mobile agent relocation, however, users can choose explicitly the mode of migration. However out method at the moment does not have a digital signature and we have a constant size of AES key length but this would be addressed by us in the future. Further improvements can be made and the network issue can be resolved during the migration by choosing the best transfer protocol for the agents. TO enhance elasticity and scalability of a multi-agent system in a trusted model we should incorporate TPM like a proxy setup.

References


agents”. In 2010 13th International Conference on Network-Based Information Systems (pp. 400-407). IEEE, September 2010.


