CHAPTER 5

CODE INTEGRITY CHECKER (CIC)

5.1 INTRODUCTION

The hardest security problem is the protection of agents from attacks coming from computational environments that are responsible for their execution. In fact, execution environments must access the agent’s code and execution state to be able to execute them. As a consequence, it is very difficult to prevent disclosure, tampering of agent parts, or incorrect execution of agents. The parts or organs of the mobile agent depicted in Figure 5.1 are subjected to security threats.

![Figure 5.1 Organs of the Mobile Agent](image)

The static part consists of the static itinerary (list of remote hosts to visit), code and data. The dynamic part consists of the execution state and the dynamic itinerary (decided by the remote hosts based on the requirements of
the agent and current conditions). The mobile agent migrating with both the static and dynamic parts may be attacked by the remote host. To protect the static and dynamic parts, the cryptographic solution and trusted environment is required. This chapter examines how to protect the mobile agent code against malicious host attacks. Chapter 6 deals with data attacks and their protection.

5.1.1 Attack Scenarios

The mobile agent migrating to the remote host is completely under the control of the remote host. The malicious remote host has the freedom to alter or destroy the agent. The reflection of the alteration is the malicious behavior of the agent in the forthcoming hosts. The altered malicious behavior affects both the originator and the forthcoming remote host.

For example, the originator initiates the agent to collect the information related to the availability of an airfare ticket from India to Australia (consider that the requirement is embedded with the agent code). Suppose that the agent visits the malicious travel agency with the requirements. Currently there are no tickets available to Australia on that date with the agency. Then the malicious agency changes the place as America instead of Australia in the agent. Then the forthcoming travel agencies have no tickets to America. Therefore the agent reaches home with the result as Nil. Then the originator has to reschedule his journey. But the air ticket is available to Australia with other travel agencies on that date. This situation creates a critical problem to the originator as well as the travel agencies.

Another example is, in an ad hoc network the client will send the agent to find the route to the destination host. The intermediate malicious host may change the behavior (agent code) of the mobile agent to shut down the forthcoming (neighboring) hosts instead of finding the route. This will create serious issues to the forthcoming hosts.
In the mobile agent code attack, the prime motive of the attacker is to attack the mobile agent and the forthcoming remote host through the agent. Figure 5.2 shows the platform to agent attack. The agent dispatched from the home to the malicious remote host is modified (attacked) and sent to the next legitimate remote host. Here, the behavior of the mobile agent is modified. Some times, the agent may be killed by the malicious remote hosts. In this case, the recovery of the mobile agent is required, which is discussed in chapter 6.

**Figure 5.2 Platform to Agent Attack**

Figure 5.3 shows the platform to agent to platform attack. The agent dispatched from the home to the malicious remote host will modify (attack) it and send it to the next remote host. The agent dispatched to the next remote host misbehaves with that and disturbs its routine process and damages its resources. In this scenario, the victim is the forthcoming host and the attacker is the innocent agent originator. The original attacker escapes from this attack.

**Figure 5.3 Platform to Agent to Platform Attack**
5.1.2 Protection Models

In the multi-hop mobile agent environment, the malicious host may perform two types of attacks on the mobile agent code, viz., passive and active attacks. In the passive attack, the malicious host can gain the copy of the agent code without its knowledge and use it for future purposes. Generally, the mobile agent program written in java is the class file (byte code). The malicious host can eavesdrop (copy) the agent class file and decompile it to get the java code and use it for malicious purposes. In the active attack (addressed in this thesis), the agent code is altered to change the behavior of the mobile agent. To protect against the alteration attack, there are two approaches, viz., preventing the attack and detecting the attack as shown in Figure 5.4.

![Figure 5.4 Protection of Mobile Agent Code](image)

(i) Prevention of Attack: An effective approach for preventing the mobile agent attack is the trusted environment (i.e., sending the mobile agent to the authenticated remote hosts). But there is no guarantee that the trusted node is always genuine. Also, the trusted model with the encryption and authentication approach to prevent the alteration attack severely hurts the concept of an open agent system when a new server joins the system (Sander and Christian 1998).
(ii) Detection of Attack: To avoid the issues of the attack prevention model, the detection of the alteration attack is the solution, and also, it avoids the long-lived attacks. In the detection method, hosts are threatened at the time of joining in the network as the host will be barred from further business if it is detected as the attacker.

To protect the mobile agent code, the ACCK Protocol (Algesheimer et al. 2001) and honest – but – curious model (Hasegawa et al. 2007) and the multiple trusted server model (Zhong and Yang 2003) used the Oblivious Transfer (OT) protocol mechanism by having the trusted third party. The drawback of the TTP model is that, trusting the sensitive or private information to a third party will always make some people uncomfortable (Ke Xu 2004).

To overcome this drawback, Ke Xu (2004) developed a multi-agent mechanism with Oblivious Transfer. Even though it provides privacy to the agent, it increases the network traffic.

Later, the self-protected mobile agent scheme (Pedro et al. 2006) was introduced which is almost the same as the code signing mechanism (Joseph et al. 1996) except for the difference in the cryptographic computation. In the self-protection scheme, the sender has to sign the agent code for integrity using the disposable key and the agent code and its signature are inserted into a digital envelope. Then, the envelope is ciphered with a symmetric key. This key is now the only part of the agent that will be deciphered with the next platform’s public key at every hop. To make the mechanism stronger, the system generates a disposable key pair to sign the agent code and to verify its integrity.
On arrival, the remote platform uses its secret key to extract the encrypted disposable key pair and the signature of the agent code. The operation will succeed if the agent’s code has not been modified. The drawback with this model is, multiple numbers of keys and their signature are used with multiple computations.

To overcome all these issues, this chapter proposes the Root Canal (RC) and eXtended Root Canal (XRC) algorithm. Also, the XRC will protect the legitimate hosts from false malicious claims.

5.2 ROOT CANAL (RC) ALGORITHM

The alteration in the agent specifies the modification of the agent code. The cruel host alteration on agent may be simple or complex but it makes changes in the behavior of the agent. Normally in the mobile agent environment, the remote host which provides the computing environment to the agent for execution doesn’t have any privileges to modify the agent code or its information, but the malicious host may breach the rule. For this, the Root Canal (RC) algorithm is proposed to identify the alteration of the code by the malicious hosts.

The Root Canal (RC) algorithm represented in Figure 5.5 consists of two modules: one is for the originator and the other one for the remote servers, all of which are receiving the agent. The function of the originator is to generate the hash value or hash code ($H_{code}$) for the agent byte code and encrypt the hash value with its private key (signature of the originator). The encrypted data is appended with the agent and dispatched to the remote hosts. The function of the remote host is to get the encrypted hash code (signature) of the agent and decrypt it using the public key of the originator. Next, it should generate the hash value for the agent code migrating from the preceding host and compare both the hash value from the originator and the
hash value generated by it. If both are the same, there is no alteration identified in the agent code. Otherwise the current availing host of the agent claims that the host which sends the agent to it has modified the agent.

Figure 5.5 Root Canal (RC) Algorithm

5.2.1 Protection against Attacks

The Root Canal algorithm will protect the agent from the multiple malicious attacks generated by the malicious hosts against the legitimate agent and legitimate host. There are different scenarios of attacks. Some typical scenarios are given below:

i) Platform to agent attack protection: The RC algorithm will protect against this attack with the use of the hash value (originator signature in the agent code). If the malicious remote host modifies the agent code and sends it to the next
remote host, then the generated hash value for the agent in the remote host and the signed hash value of the originator will differ. If the succeeding remote host finds any alteration in the agent code then it will abort the agent function. The signature of the originator is verified with the help of the public key \( <Pb_0> \) available with the remote hosts.

ii) **Platform-agent-platform attack Protection:** The mobile agent migrating from any host should be verified for any attack before allowing it to process. The verification of the agent is to identify whether the agent is having any malicious activity or the agent behavior (code) is modified by the preceding host. The identification of the malicious activity of the agent is already described in the previous chapter, with the scanning mechanism. To identify the alteration of the agent code in the middle of the cruise, the signature of the originator should be verified. It means that, the host currently having the agent also has the public key of the originator to verify the signature by generating the hash code for the received agent code from the preceding host. Suppose the generated and received hash code (from the originator) does not match; then, there is an alteration and it is identified and intimated to the originator and administrator for further action (assumption to block the host for some time or permanently). If there is no alteration in the agent code then the agent is allowed to continue its process.

iii) **Insertion Attack protection:** In the multi-hop mobile agent environment it is possible for the malicious host to insert an additional code to alter the behavior of the agent. The insertion of the code is also identified by the RC algorithm.
5.2.2 Malicious Host Claim

The host receiving the mobile agent can have the rights to claim for malicious modification of the agent code. As the succeeding host identifies the malicious changes in the agent code received from the preceding host, it has the right to claim for the malicious change against the preceding host. In some situations, the malicious host can claim for the malicious against the legitimate preceding host. Explicitly, the mobile agent migrating from the legitimate host to the malicious host is verified for any alteration. The malicious host will suddenly claim for the malicious against the legitimate preceding host. Sometimes, the malicious succeeding host will modify the agent code and claim for the malicious change by the preceding host. For this situation, the proposed Root Canal (RC) algorithm is not fit for the effective security measures. To protect this type of false claim, the RC algorithm needs to be extended.

5.3 EXTENDED ROOT CANAL (XRC) ALGORITHM

To protect against the false malicious claim issue, the Root Canal algorithm is extended with the digital signature of the every host. It is mainly for non-repudiation purposes (i.e., the sender of the data is not able to claim that the data is not mine). The malicious host is not able to claim for a change in the agent code without having the complete proof. The proof is the signature on the encrypted hash code by the previous host. The processing steps of the extended root canal algorithm (XRC) are given in Figure 5.6.
<Originator>
<Start>
Generate the hash code <H<sub>code</sub>> for the agent byte code
Encrypt the <H<sub>code</sub>> using the private key <Pr<sub>0</sub>> and get <Ê<sub>U</sub>>
Dispatch the agent <Ê<sub>Ê</sub>>

<First Remote Host>

Generate the hash code <RH<sub>code</sub>> for the received byte code
Decrypt the code <Ê<sub>U</sub>> using the public key <Pb<sub>0</sub>> and extract <H<sub>code</sub>>
if (<H<sub>code</sub>> = <RH<sub>code</sub>>) then
{
Allow the agent to perform its task <Ê<sub>Ê</sub>>
Digital signature Ŷ<sub>0</sub> for <Ê<sub>U</sub>> using the private key <Pr<sub>1</sub>>
}
else
{
Claim for the malicious host
}
<End>

<Every other Remote Host>

Verifies the sign Ŷ<sub>0</sub> of <Ê<sub>U</sub>> using the public key <Pb<sub>i-1</sub>> after Hashing <Ê<sub>U</sub>>
Generate the hash code <RH<sub>code</sub>> for the received byte code
Decrypt the code <Ê<sub>U</sub>> using the public key <Pb<sub>0</sub>> and extract <H<sub>code</sub>>
if (<H<sub>code</sub>> = <RH<sub>code</sub>>) then
{
Allow the agent to perform its task <Ê<sub>Ê</sub>>
Digital signature Ŷ<sub>0</sub> for <Ê<sub>U</sub>> using the private key <Pr<sub>i</sub>>
}
else
{
Claim for the malicious host
}
<End>

Figure 5.6 eXtended Root Canal (XRC) Algorithm

Initially, the originator has to sign the agent code by taking the hash value and encrypt it with the private key. The signed data is added with the agent and sent to the first remote host. After the arrival of the agent, the first remote host will verify the signature of the originator and allow it for
execution. At the end, the remote host has to sign the encrypted hash value (signature of the originator) of the agent and dispatch the agent to the next remote host. The second remote hosts which receive the mobile agent have to verify the signature of the preceding host first and proceed with another verification function to identify the alteration of the agent code. After the verification and agent computation, the current host should remove the signature of the preceding host on the encrypted hash code and put its own signature on that. This will be continued until the journey comes to an end. If any malicious host raises a false claim against the legitimate host, it can easily be identified by the signature.

5.4 EXPERIMENTAL RESULTS

5.4.1 Experimental Setup

The proposed RC and XRC algorithm is implemented and tested in the systems with the configuration of 1GB RAM and 2.2 GHz Dual processor. The IBM Aglet is the agent server used to accommodate the agent. The IBM aglet is best for the mobile agent migration and communication (it supports interoperability and open source). The SHA-1 (if necessary we can use SHA-256 or SHA-512) algorithm is used to generate the hash value for the offer of the agent byte code. The RSA algorithm is used for the encryption and it will not encrypt the non-numerical data. For that reason, there is a need to convert the non-numerical data into numerical data. Every host in the environment should use the same conversion mechanism to convert the numerical into non-numerical data and vice versa.

5.4.2 Results of RC Algorithm

The Root Canal algorithm is implemented and tested between the two systems for multiple times. The measured time of the agent at the
originator is the time taken to sign the mobile agent code and time for the remote host is includes the signature verification time and agent processing time. Table 5.1 shows the time to perform the computation in the agent home and the time taken by the remote host for verification and computation of data.

Table 5.1  RC Algorithm Processing Time (in milli seconds)

<table>
<thead>
<tr>
<th>Executions</th>
<th>Processing Time at Agent Home (AH)</th>
<th>Processing Time at Remote Host (RH\textsubscript{1})</th>
<th>(\sum\text{(AH, RH\textsubscript{1})})</th>
</tr>
</thead>
<tbody>
<tr>
<td>E\textsubscript{1}</td>
<td>93</td>
<td>797</td>
<td>890</td>
</tr>
<tr>
<td>E\textsubscript{2}</td>
<td>94</td>
<td>656</td>
<td>750</td>
</tr>
<tr>
<td>E\textsubscript{3}</td>
<td>94</td>
<td>750</td>
<td>844</td>
</tr>
<tr>
<td>E\textsubscript{4}</td>
<td>79</td>
<td>790</td>
<td>869</td>
</tr>
<tr>
<td>E\textsubscript{5}</td>
<td>93</td>
<td>766</td>
<td>859</td>
</tr>
<tr>
<td>E\textsubscript{6}</td>
<td>94</td>
<td>844</td>
<td>938</td>
</tr>
</tbody>
</table>

The RC algorithm is executed (E) for six times in the same configured environment to find the average execution time of the algorithm. The values in the Table 5.1 vary for each execution, as the multiple processes is running parallel in the same machine (every remote host has multiple processes). Hence the average for the six executions is taken. It is not only applicable for this table; it is also for all the tables given in this thesis. The average execution time of the algorithm in the home and the remote host is as follows:

Average time to execute the RC at Home = \(\frac{\sum_{i=1}^{6} E_i}{6}\) = 91 ms \hspace{1cm} (5.1)

Average time to execute the RC at RH = \(\frac{\sum_{i=1}^{6} E_i}{6}\) = 767 ms \hspace{1cm} (5.2)
Generally, the agent running in the aglet tahiti server is the byte code (java class file). The byte code in the aglet Tahiti server and the same byte code in the host will have small changes (even though the byte code gets small change, the Root Canal algorithm will not allow the agent to execute). The snapshot of Figure 5.7 shows one of the changes in the byte code of the host and the byte code got from the aglet server. Therefore, the hash code generated for the byte code should be taken only from the aglet server and not from the place where the byte code is stored in the host.

![Snapshot of the Byte Code Variation](image)

5.4.3 Results of the XRC Algorithm

To avoid a malicious claim in the mobile agent environment, the RC algorithm is modified as the eXtended Root Canal (XRC) algorithm. It includes the additional digital signature on the remote host side. Every host
receiving the mobile agent should have to sign the encrypted hash code of the agent from the originator before dispatching it to the next host.

Table 5.2  XRC Algorithm Processing Time (in milli seconds)

<table>
<thead>
<tr>
<th>Executions</th>
<th>Processing Time at Agent Home (AH)</th>
<th>Processing Time at Remote Host (RH₁)</th>
<th>( \sum (AH, RH₁) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>E₁</td>
<td>93</td>
<td>829</td>
<td>922</td>
</tr>
<tr>
<td>E₂</td>
<td>94</td>
<td>843</td>
<td>937</td>
</tr>
<tr>
<td>E₃</td>
<td>94</td>
<td>844</td>
<td>938</td>
</tr>
<tr>
<td>E₄</td>
<td>79</td>
<td>812</td>
<td>891</td>
</tr>
<tr>
<td>E₅</td>
<td>93</td>
<td>803</td>
<td>896</td>
</tr>
<tr>
<td>E₆</td>
<td>94</td>
<td>843</td>
<td>937</td>
</tr>
</tbody>
</table>

Table 5.2 gives the processing time of the XRC algorithm in the agent home and the remote hosts. The agent home includes only the signing time and the remote host includes the signature verification time and newly added signing time. The execution time is varying from one execution to another because of the parallel process in the hosts. The XRC algorithm is executed six times in the same configured environment to find the average execution time of the algorithm. The average execution time of the algorithm in the home and the remote host is as follows:

\[
\text{Average time to execute (E) the XRC at AH} = \frac{\sum_{i=1}^{6} E_i}{6} = 91 \text{ ms} \quad (5.3)
\]

\[
\text{Average time to execute (E) the XRC at Remote Host (RH₁)} = \frac{\sum_{i=1}^{6} E_i}{6} = 829 \text{ ms} \quad (5.4)
\]
5.4.4 Comparison of RC and XRC Algorithm

The difference between the RC Algorithm and XRC Algorithm is the additional digital signature to avoid the false claim by the malicious host. Table 5.3 gives the processing time difference between the two algorithms in the agent home and remote host side (very first remote host for the agent from the originator).

Table 5.3 Processing Time Difference between RC and XRC Algorithm in RH₁ (in milli seconds)

<table>
<thead>
<tr>
<th>Executions</th>
<th>Agent at Home</th>
<th></th>
<th>Agent at Remote Host (RH₁)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RC</td>
<td>XRC</td>
<td>RC</td>
<td>XRC</td>
</tr>
<tr>
<td>E₁</td>
<td>93</td>
<td>93</td>
<td>797</td>
<td>829</td>
</tr>
<tr>
<td>E₂</td>
<td>94</td>
<td>94</td>
<td>656</td>
<td>843</td>
</tr>
<tr>
<td>E₃</td>
<td>94</td>
<td>94</td>
<td>750</td>
<td>844</td>
</tr>
<tr>
<td>E₄</td>
<td>79</td>
<td>79</td>
<td>790</td>
<td>812</td>
</tr>
<tr>
<td>E₅</td>
<td>93</td>
<td>93</td>
<td>766</td>
<td>803</td>
</tr>
<tr>
<td>E₆</td>
<td>94</td>
<td>94</td>
<td>844</td>
<td>843</td>
</tr>
<tr>
<td>Avg.</td>
<td>91</td>
<td>91</td>
<td>767</td>
<td>829</td>
</tr>
</tbody>
</table>

The processing time in the agent home for both the RC and XRC algorithms is the same because the same process is followed for both algorithms but the processing time of the agent in remote host differs. Even though the XRC takes more time than the RC algorithm, it is able to protect all types of attacks (code alteration and false claim) efficiently.

5.5 COMPARISON WITH THE EXISTING SCHEME

The self-protected mobile agent (Pedro et al 2006) for e-health environment has its own agent code protection mechanism. The solution of
protecting the mobile agent is based on a public-decryption function provided by the platform through a cryptographic service which is accessed by properly structured agents. The proposed RC and XRC algorithms are also the same as the self protection scheme (Pedro et al 2006) with the less overhead of the encryption function as regards both size and time. Table 5.4 gives the computational complexity comparison of each model to check the integrity of the agent code. The self-protection scheme requires some additional computation cost on both the sender side and the receiver side than the Root Canal (RC) and the eXtended Root Canal (XRC) algorithms.

Table 5.4  Computational Complexity Comparison of Existing and Proposed Algorithms

<table>
<thead>
<tr>
<th>Place of Agent</th>
<th>Existing Scheme</th>
<th>Proposed Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Self-Protection Scheme</td>
<td>RC</td>
</tr>
<tr>
<td>Agent at Sender</td>
<td>t\textsubscript{SA} + t\textsubscript{EAS} + t\textsubscript{HC} + t\textsubscript{ESC}</td>
<td>t\textsubscript{SA} // only in case of agent home</td>
</tr>
<tr>
<td>Agent at Receiver</td>
<td>t\textsubscript{DAS} + t\textsubscript{DSC} + t\textsubscript{VC} + t\textsubscript{VS}</td>
<td>t\textsubscript{VS}</td>
</tr>
</tbody>
</table>

Table 5.5 shows the time comparison of the self-protection scheme and the proposed Root Canal (RC) and the eXtended Root Canal (XRC) algorithms in the first remote host. The execution time in the Table 5.5 are less when compared to that in Tables 5.1 to 5.3 because of the modification in the implementation (to store the encrypted value, a variable is used here instead of inserting into and retrieving from the database). The comparison in
Table 5.5 shows that the proposed model is more efficient than the existing model by saving the processing time of 67% and 33%.

### Table 5.5  Computational Time Difference between Existing Scheme and Proposed Algorithms in RH<sub>1</sub> (in milli seconds)

<table>
<thead>
<tr>
<th>Executions</th>
<th>Existing Self-Protection Scheme (a)</th>
<th>Proposed Models</th>
<th>Proposed Models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Execution Time (b)</td>
<td>% of Saving (a-b)/a</td>
<td>Execution Time (c)</td>
</tr>
<tr>
<td>E&lt;sub&gt;1&lt;/sub&gt;</td>
<td>47</td>
<td>15</td>
<td>68.1</td>
</tr>
<tr>
<td>E&lt;sub&gt;2&lt;/sub&gt;</td>
<td>47</td>
<td>15</td>
<td>68.1</td>
</tr>
<tr>
<td>E&lt;sub&gt;3&lt;/sub&gt;</td>
<td>46</td>
<td>16</td>
<td>65.2</td>
</tr>
<tr>
<td>E&lt;sub&gt;4&lt;/sub&gt;</td>
<td>47</td>
<td>15</td>
<td>68.1</td>
</tr>
<tr>
<td>E&lt;sub&gt;5&lt;/sub&gt;</td>
<td>48</td>
<td>16</td>
<td>66.7</td>
</tr>
<tr>
<td>E&lt;sub&gt;6&lt;/sub&gt;</td>
<td>48</td>
<td>15</td>
<td>68.7</td>
</tr>
</tbody>
</table>

In addition to the computational cost, the self-protection scheme requires the additional memory cost of the agent. The additional memory cost of the agent in the self protection scheme is the control code, encryption form of the symmetric and asymmetric key. The agent size \(Q_{A\text{code}}\) in the proposed RC and XRC is always less than the existing self-protection scheme (Pedro et al 2006) by having the encrypted agent code with multiple keys. The RC and XRC algorithms have only the normal agent code with the encrypted hash code (size of encrypted hash code is, \(Q_{H\text{code}}\)) but in the case of the XRC, the additional one more encrypted hash code (signature of the current host) (size of encrypted hash code \(Q_{AH\text{code}}\)) is added to avoid the false malicious claim. In self-protection scheme, apart from the agent code size and hash code size, the additional size of hash code of keys \(Q_{KH\text{code}}\) and the three keys
(size of 3 key is, $Q3key$) are added. The agent size for the different algorithm and scheme are calculated as follows:

Proposed RC Algorithm = $Q_{A_{code}} + Q_{H_{code}}$

for 8b key = $(2KB=2*1024B) = 2048 + 20 = 2068$ \hspace{1cm} (5.5)

Proposed XRC Algorithm = $Q_{A_{code}} + Q_{H_{code}} + Q_{AH_{code}}$

for 8b key = $(2KB=2*1024B) = 2048 + 20 + 20 = 2088$ \hspace{1cm} (5.6)

Existing Self-Protection scheme = $Q_{A_{code}} + Q_{H_{code}} + Q_{KH_{code}} + Q3key$

for 8b key = $(2KB=2*1024B) = 2048 + 20 + 20 + 24 = 2091$ \hspace{1cm} (5.7)

The SHA-1 algorithm will produce 160 bit hash value which is equivalent to 20B and in the self-protection scheme, the value 24 refers to the size of the three keys (disposable private and public key and the additional symmetric key).

Table 5.6 gives the difference between the proposed and existing model, regarding the agent size. The proposed RC algorithm saves the agent size of disposable key hash value and three additional keys and the XRC algorithm saves the agent size of the three key values than the existing self protection scheme. The result shows that the proposed models are better than the existing model regarding the size, computational time and protecting the agent code (both for malicious alteration in the middle of the journey and the false malicious claim). Apart from the Self-Protection Scheme, MARISM-A (Robles 2002) the agent platform protects the agent code through hashing and signature. In MARISM-A, the additional hashing requires an additional 20 bytes of agent size than the proposed algorithms, and also, it is not capable to protect against the false malicious claim.
Table 5.6  Comparison of RC and XRC Algorithms with the Existing Scheme

<table>
<thead>
<tr>
<th>Key Size (in bits)</th>
<th>Existing Self-Protection Scheme (Agent size in Bytes)</th>
<th>Proposed Models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(a)</td>
<td>Agent size in Bytes (b)</td>
</tr>
<tr>
<td>8</td>
<td>2091</td>
<td>2068</td>
</tr>
<tr>
<td>16</td>
<td>2094</td>
<td>2068</td>
</tr>
<tr>
<td>32</td>
<td>2100</td>
<td>2068</td>
</tr>
<tr>
<td>64</td>
<td>2112</td>
<td>2068</td>
</tr>
</tbody>
</table>

5.6  SUMMARY

The protection scheme for the agent code is modeled in the aspect of cipher computation. The proposed Root Canal algorithm (RC) is to protect the agent code against the malicious attacks. But, it is not able to protect the false claim by the malicious host. To overcome this false malicious claim issue, the eXtended Root Canal (RC) algorithm is introduced with the additional signature scheme.

The originator or remote host can abort the agent execution when it is identified that the mobile agent code is altered by the preceding host. In the colluded host attack, the immediate preceding host of the legitimate remote host is considered as the attacker. In comparison with the existing scheme as regards computation and agent size, it illustrates that the proposed Root Canal (RC) algorithm is more efficient for protecting the mobile agent code.