Protecting the Data State of Mobile Agents by Using Bitmaps and XOR Operators

Jesús Arturo Pérez Díaz,
ITEMS, Campus Cuervanava
Av. Paseo de la Reforma 182-A, 62589, Temixco, Morelos, Mexico.
jesus.arturo.perez@itesm.mx
AND
Dario Álvarez Gutiérrez,
University of Oviedo
Calvo Sotelo s/n, 33007 Oviedo, Spain.
darioa@pinon.ccu.uniovi.es

Keywords: security, mobile agents, data state protection, encryption

Received: June 20, 2001

Mobile agents have been considered a promising technology to develop e-commerce applications, however the security concerns about the technology have stopped their widespread use.

The identified security areas comprise protecting hosts against malicious agents, protecting the agent’s transmission and protecting agents against malicious hosts. The first two security issues and the protection of the agent’s code state can be solved by applying traditional security techniques. Even though there are some works that manage the privacy of execution, their implementation is almost unfeasible in terms of performance and complexity.

This paper describes a fast and easy to implement algorithm that a mobile agent can use to encrypt its data during its itinerary. The algorithm only makes use of a bitmap and XOR operations. The algorithm consist of applying XOR operations to the data to be ciphered and a random bitmap, while the map is repeatedly shifted to the right or to the left in order to compute a CRC field for validation against malicious tampering.

The method only uses basic bit operations so that its implementation is very easy to develop. Besides, since it does not use any computationally expensive cryptographic technique (i.e. digital signatures) it is very fast. In this way we manage to have a secure, simple, fast and feasible protection algorithm to protect data while mobile agents are roaming, where simplicity and performance are its better advantages.

1 Introduction

Mobile Agent Systems are expected to make e-commerce transactions inside virtual supermarkets. In this application area security is crucial since we can consider that any application will not be useful without doing secure transactions.

Mobile agents consist of code state, data state, and execution state. Mobile agent systems are platforms that allow agents to migrate from one node (a mobile agent system) to another, keeping its three states. While agents migrate there are several security aspects involved. We can point out different mechanisms that must be implemented by the mobile agent system to ensure the security of the mobile agent applications. Mobile agent systems basically must provide:

- Protection of the agent system against attacks from mobile agents.
- Protection of the agent against other agents.
- Protection of information transmission between agent servers against unauthorized third parties.
- Protection of the agent against malicious agent systems (malicious hosts), which includes protection of the data state of the agent.

Different security architectures for mobile agents [1] and mobile agent systems [2] [3] have used standard cryptographic techniques like public key cryptography, or digital signatures to authenticate authorities and solve the problem of protecting the host against malicious agents. Also, they have implemented secure channels for the transmission of the agents by using SSL or TLS. Nevertheless, the protection of mobile agents from malicious hosts is only partially solved.
The code state of the agent can be signed since it will not be modifiable. In this way, we can protect the static part of the agent. However, to protect the data state (that changes dynamically) becomes a more difficult task to tackle.

There are some works in this area, described in the related work section. However, there has not been found any solution having a feasible implementation.

Consider that most of mobile agent e-commerce applications do not need to protect all the data state, but only some important values where agents filter information and compile their results. The algorithm presented in this paper protects all the data that the agent decides to encrypt by calling a cipher function. When the agent returns, only the source server is able to decrypt the sensible information stored by the agent.

The algorithm describes an easy way to protect sensible data that must be gathered and are carried by mobile agents alongside their itinerary. Two principal advantages are highlighted: the algorithm is simple and feasible to implement, and computationally inexpensive.

## 2 Related Work

Wilhelm presented a technique for protecting the itinerary of the mobile agents by using hardware mechanisms [4]. He considered that software algorithms were not enough to ensure complete security during the mobile agent’s itinerary. Even though the technique managed to achieve the protection of the itinerary, its implementation in real applications becomes difficult, since special hardware is required.

One interesting approach to avoid the malicious host attacks was proposed by Fritz Hohl [5]. This approach, which is called Code Mess Up, consists of a combination of two mechanisms: the first one generates a new and far less understandable version of the agent’s code. The second mechanism restricts the lifetime of the agent’s code and data. In this way, when the code of the agent is messed up, the malicious server would take some more time in order to understand the code and then attack it, but since the agent’s lifetime is restricted, the malicious server will not have enough time to attack the agent. In this way the agent remains untouched.

Another solution for this problem was proposed by Tomas Sander and Christian Tschudin [6][7]. They presented techniques on how to achieve “non-interactive computing with encrypted programs” in certain cases and give a complete solution for this problem in important instances. They further show how an agent might securely perform a cryptographic primitive, digital signing, in an untrusted execution environment. Their results are based on the use of homomorphic encryption schemes and function composition techniques.

The last two solutions were designed to offer privacy on the agent’s execution, but not to give privacy and integrity to the agent’s data. Beside, both of them have two main problems: a quite difficult implementation and a considerable performance hit in case of implementation. Perhaps these disadvantages are the reason why these techniques have not been implemented by any mobile agent system, as far as we know, and the problem in current mobile agent systems is still unsolved.

Considering these related works, the goal of our research is to offer a simple and feasible to implement algorithm that can be used by mobile agents just for encrypting the data that they gather while they are roaming in untrusted execution environments, and without a perceivable performance hit.

## 3 Data Encryption Using Bitmaps and the XOR Operation

The design of this data protection technique takes into account the fact that, in most applications, it is not vital to protect the whole data state of the mobile agent but some variables holding sensible data gathered by the agent, which is the main goal of the agent’s travel and need to be protected.

Typical examples of these applications are e-commerce applications in which an agent travels alongside an itinerary looking for prices or particular services. The vital information that must be protected is the price or service offered in each visited server.

This technique requires that the agent travels holding data generated by the source server that will be used by the agent to encrypt the sensible data gathered, using fast XOR operations.

### 3.1 Usefulness of the XOR Operator

The main encryption idea is to apply the XOR operation between data and a random number (expressed as a bitmap in a row of a matrix and known only by the source server) to encrypt information. Once the agent returns to the source server, the XOR operation is applied again to the encrypted data, using the same random number, and the information is restored.

The agent in the source server will generate two matrixes with a number of rows equal to the number of data items it expects to encrypt. Initially, both matrixes will be filled with the same random numbers (forming a random background bitmap). One of the matrixes will be stored in the source server and the agent will carry the other.

For example, let’s assume a 10 (binary 1010) is generated as a random number and put it in a row of the matrix. This number goes with the agent and a copy is also stored in the source server. During the itinerary, the agent gets a 3 (binary 0011) that the agent wishes to encrypt. The XOR operation will be then applied between the random number and the datum to be protected (1010 ⊕ 0011), giving 1001 as a result. This
The structure of the matrix is as follows: each datum is stored in the same row of the matrix, overwriting the initial random mask, in order to avoid the next server seeing the random number used to encrypt the datum. Also, the next server is not able to know the real datum, since it ignores the random number used to apply the XOR operation.

A given server will use the next free row available in the matrix to store new data, as the occupied rows contain data encrypted in previous servers. In this way, the current server will never be able to know the previously encrypted data since it does know neither the datum nor the random number.

The source server had stored in a duplicate matrix a copy of the random numbers, in order to retrieve when the agent returns, the data encrypted by the agent while roaming from server to server. Thus, to retrieve the datum, the encrypted number 1001 will get a XOR applied with the corresponding random number generated in the source server (1001 XOR 1010), giving 0011 as a result (3, the datum the agent had encrypted).

To assure that the information restored upon return to the source server has not been tampered with, and is the same information that the agent encrypted in each server, a CRC field is computed in order to perform an integrity test.

The complete encryption algorithm is described in the next section.

### 3.2 Detailed Description of the Encryption Algorithm

A matrix with several fields is defined (table 1), which is used to encrypt the agent’s data and for validation of the data later. The matrix is initially filled with random numbers, creating a background bitmap used to encrypt data gathered by the agent alongside its itinerary (table 2). The matrix is duplicated. One copy travels with the agent and the other is kept in the source server. The source server’s matrix is used to recover the data upon agent’s return.

Every datum to be protected by the agent needs a row of the matrix, so the agent must know beforehand the approximate amount of data it is going to use.

The structure of the matrix is as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host ID</td>
<td>The identifier of the server, as we need to know, for each row, the place where data was encrypted.</td>
</tr>
<tr>
<td>Data to be protected</td>
<td>The second field represents the space needed to store the data to be gathered by the agent, in 128 bit blocks.</td>
</tr>
<tr>
<td>CW</td>
<td>The third field is the “codeword”, which is a random number to be generated in the remote server.</td>
</tr>
<tr>
<td>CRC</td>
<td>The CRC field is computed as follows.</td>
</tr>
</tbody>
</table>

The first field is the identifier of the server, as we need to know, for each row, the place where data was encrypted. The second field represents the space needed to store the data to be gathered by the agent, in 128 bit blocks. The third field is the “codeword”, which is a random number to be generated in the remote server. The codeword is used to rotate data before applying the encryption function. The last field is a CRC, which is computed applying a XOR operation using all the 128 bit blocks in the data area. This CRC is used upon agent return to verify that the data area has not been altered.

### Table 1. Fields composing the rows of the matrix

<table>
<thead>
<tr>
<th>Host ID</th>
<th>Data to be protected</th>
<th>CW</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>fi</td>
<td>fi</td>
<td>fi</td>
<td>fi</td>
</tr>
</tbody>
</table>

When the agent departs from the source server, the agent carries the matrix filled with random numbers, creating a background bitmap that is used to hide information, as shown in the next table:

### Table 2. Matrix filled with a random generated bitmap

<table>
<thead>
<tr>
<th>ID Host</th>
<th>Data Area of the Mobile Agent</th>
<th>CW</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>128 bits</td>
<td>128 bits</td>
<td>128 bits</td>
</tr>
<tr>
<td>0101</td>
<td>1101 0011 0101 0100 0100</td>
<td>0100 1101 0101 0011 0100</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>0001 1101 010 0001 0100 0100</td>
<td>0100 1101 0101 0011 0100</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>0011 0101 001 1101 0100 1101</td>
<td>0100 1101 0101 0011 0100</td>
<td></td>
</tr>
</tbody>
</table>

Alongside the itinerary, the following algorithm is applied for each datum to be encrypted:

1. The remote server creates a record with the same fields than a row of the matrix that the agent has.
2. The host ID, data to be encrypted in 128-bit blocks form, and a generated random codeword (CW) are put into the record.
3. Each 128 bit block fi is rotated to the left as many times as indicated by the 7 less-significant bits of the CW. That is fi \( \Leftarrow fi << li \), where li \( \Leftarrow CW & 07Fh \).
4. Before applying the third step on \( fi+1 \), the CW is rotated to the right as many times as indicated by the 7 most-significant bits of the CW. Thus, the number of times that each fi is rotated is not always the same. CW will then be \( CW \Leftarrow CW >> mi \) where mi \( \Leftarrow (CW << 7) & 07Fh \). Once the CW is rotated step 3 is repeated. These tasks will continue until no more 128-bit blocks are left.
5. The original CW is restored into the corresponding field of the register in order to retrieve the original information using the inverse algorithm in the source server.
6. The CRC field is computed as follows. The initial value is filled with binary 0’s, and then it is XOR’ed in sequence (from left to right) with all the 128 bit blocks in the data area, giving the final CRC value.
7. Lastly, the corresponding row in the matrix holding the original bitmap is XOR’ed with the generated register (data) to be protected, thus encrypting the data.

8. The counter indicating the number of lines used in the matrix is incremented so that the next row available of the matrix can be used.

It is worth to note that, once the mobile agents arrive at a new server, the new server can not access the information stored in the previous server, as the background bitmap held in the previous row before the XOR operation was applied can not be guessed. Only the source server, who has a copy of the original matrix, is able to apply the inverse algorithm to retrieve the encrypted data.

The CRC field is computed in order to detect any alteration made by a malicious server on the encrypted data. Without any validation action, a malicious server could modify just one bit on the encrypted data field and when the agent returns to its source server it would not be able to detect that alteration. So, the agent would recover a wrong modified value since the source server just applies and XOR operation with the stored copy of the matrix in order to get the encrypted value.

The CRC field does not prevent a malicious server from making any alteration, but it ensures that if an alteration were made it would be detected since the CRC field will be invalid.

The bit rotations made in step 3 may appear unnecessary. However, if the blocks are not rotated, a malicious server could alter the encrypted information in just one bit in a specific position, and the CRC may not change since each block is XORRed with the next block. Then, upon return to the source server, this alteration would not be detected.

On the other hand, once the random rotations are applied, a maliciously-altered bit in one block would be detected, as this bit affects many positions in the inverse decryption algorithm (since the position of that bit will change after rotations are applied), rendering always an invalid CRC that will detect the alteration.

To retrieve the information encrypted by the agent alongside the itinerary, the source server just applies the XOR operation to each row of the matrix that was used by the agent, with the corresponding row of the copy of the original matrix holding the initial background random bitmap.

Then, using the random CW, inverse rotations are applied to retrieve the real data that was encrypted by the agent in a given intermediate server.

The main advantage of this technique, encrypting data using bitmaps and XOR operations is that is very easy to implement, compared with other methods, which use very complex mathematical algorithms. Besides, it is computationally inexpensive, as only very fast bit operations are used, avoiding effectively the performance impact of other techniques such as digital signatures, keys, or any other means that hurt performance.

4 Feasibility of Implementation and Incorporation to Current Mobile Agent Systems

A great advantage of our protection scheme is the feasibility of implementation. Besides, it could be very easily incorporated to the current mobile agent systems' security mechanisms.

The majority of Java-based mobile agent systems define an abstract class called Agent. All the agents programmed by the user inherit from this class the required functionality, so that the agent can migrate from one host to another or can create more agents. This abstract class usually follows a pattern like this:

```java
public abstract class Agent implements java.io.Serializable {
    public void run()
    public final java.lang.Object clone()
    public final void createAgent(.....)
    public final void dispatch(java.net.destinationURL)
    public final void revert()
    ...
}
```

We just need to add an addsecure() method to the agent abstract class in order to allow the agents to securely store sensible information in the data structure (the matrix of bitmaps) that is carried with them, so we could define:

```java
public final void addsecure(Object data)
```

The implementation of this method will encrypt the information using the algorithm described in the previous section and will store it in the next row available of the matrix that is carried with the agent.

The matrix can be easily defined in Java using a Java array (i.e. an instance of the class vector) that will hold the background bit map originated in the source server.

In this way, each time that an agent is created by a user (i.e. commerceAgent), it will inherit the addsecure() method allowing it store information in a secure way.

```java
public class commerceAgent extends Agent {
    ....
}
```

When the user creates an instance of commerceAgent, the instance will be able to protect the information it is
gathering, just by invoking the addsecure() method. For example:

```java
commerceAgent findFlyAgent;
```
defines an agent of type commerceAgent. The run method of the findFlyAgent would contain the instructions to query the price of the fly it is looking for, once it gets the price it records for later analysis at the home server, so the last line of the run method would be:

```java
findFlyAgent.addsecure(FlyPrice);
```
in order to protect the sensible datum it has gotten. The current server will execute the encrypting process and will store the FlyPrice safely in the matrix. The next server visited will not be able to find out what was the price in the previous one and it just will be able to encrypt the information that the agent gathers in that server.

5 Limitations of the Method

The algorithm allows to protect the information the agent decides during its itinerary, and to verify that it has not been altered when the agent returns. The algorithm does not prevent the possible alteration of data from malicious hosts, but detects any modification that has been made. In this way, if any alteration is detected (which means a CRC field is invalid) the agent will reject the information since it would be considered invalid. In this way our technique offers integrity.

The current server will never be able to access the previously encrypted data since it ignores the data and the random number used to apply the XOR operation. However, it can see and copy the still available rows with random numbers that will be used to encrypt the next data not only in the current server but in the next server as well.

The first, and most evident deriving of this, is that a visited server cannot retrieve the data that was encrypted before, but could easily make a copy of the rest of the background bitmap. This means that a server could potentially retrieve the data encrypted in the future by an agent, assuming that the agent visits again the same server. Thus, an agent should not visit the same server twice if it wants to be completely secured.

Another possible attack (although less probable) is that two cooperating malicious servers teamed to retrieve the information carried by the agent. The first server would send to the second one a copy of the unused part of the background bitmap already known by the first server (the available rows of the matrix). If the agent arrived later to the second malicious server, it would be able to retrieve the data encrypted since the agent left the first malicious server and then modify the values.

The last limitation is that there is a fixed maximum number of data that can be protected, which is given by the length of the matrix (the length must be set in advance). However, in practice, a reasonable length could be set, according with the expected task to be carried by the agent.

Finally, this technique does only protect the part of the data state of the agent that the agent wishes to encrypt. The rest of the data, such as local variables, etc. are not protected.

6 Future Work

We will continue working on this technique in order to implement an improved algorithm that avoids the current limitations. Nevertheless we intend to use only the operations included in this paper (bit rotations and XOR operation), or equally fast or simple ones, so that we can keep its simplicity and fast speed which are the objectives and philosophy of this work.

7 Conclusions

One of the problems that a mobile agent system must solve is the protection of agents from malicious hosts, which includes the protection of the data state of an agent. This is very important in order agent technology be adopted in e-commerce applications, for example in applications where agents collect information (such as flight prices) for later analysis at the source server.

Protecting this data gathered by the agent (and not the whole data state, which is not vital) is the objective of the research described in this paper. For example, malicious servers should not be able to see or modify the information gathered in order to change previous low prices to make its price appear as the best.

Other techniques such as [5] and [6] try to solve the problem by privacy of execution applying very complex techniques, which are very difficult to implement, and, more importantly, are very expensive computationally, as key cryptography is used. This is a hurdle very difficult to overcome in practical systems.

We propose a new technique to protect the part of the data state of an agent (the data gathered the agent wishes to protect) that does not suffer from these limitations, as it is fast and easy to implement.

A matrix is generated at the source server, and filled initially with random bit numbers. Each row is used to protect one datum, and is divided into 128 bit blocks. A copy of the matrix is stored at the source server, and the other copy travels with the agent.

To protect one data item, the agent uses a row, and applies the XOR operation to the data item with the random number held previously, encrypting it and overwriting the initial random number with the result. A CRC is computed using XOR operations also, in order to detect alterations when returning to the source server. Some bit alterations would not change the CRC, so the random bitmaps are bit-rotated n-times, as indicated by a random codeword that is also held in the matrix (and is also rotated).
The original server is the only one able to decrypt the information, since the inverse algorithm (basically undoing the rotations and applying the XOR operation with the original random bit map) requires the knowledge of the original random bit map and codewords, which is only known (the bitmaps) by the original server.

The only limitation is that an agent should not visit the same server twice, or a server co-allied with a malicious server, as a copy of the matrix could be made and subsequent encrypted data items could be retrieved. A minor limitation is that the agent should estimate the maximum number of data items to protect, as the matrix must be generated beforehand.

The technique we have presented removes the complexity and computational limitations of other techniques, which hinder the acceptance of agent technology in real applications. Agent’s data state protection is made feasible in practical applications, as no performance hit is introduced because no expensive key cryptography is used. Furthermore, the algorithm could be a lot of times faster than any other that uses traditional key cryptographic techniques since only bits operations are used.

This algorithm can be easily integrated in current mobile agent systems in order to create basic e-commerce applications that compile information securely.

References


