Abstract - Mobile agent technology is a new paradigm of distributed computing that can replace the conventional client-server model. Protection of Mobile agents is one of the most difficult problems in the area of mobile agents security. The existing solutions that provide complete protection of agents against malicious hosts either only detect or to some extent prevent attacks on agents. In this paper, we discuss our implementation of one of the security approach that implements homomorphic encryption on a part of the data used in the Aglet environment to be executed on the hosts without decrypting it. The main idea is to use central host as a server which is responsible of encrypting and decrypting operations. The central host will encrypt some parts of data doesn't need to be known from other hosts applied on finding the longest path of directed graph problem. Other hosts only add their data to decrypted parts and pass it to the central host to perform the full operation there. We get the result of the problem printed on the console window securely and protected from hosts' attacks. This security approach produce a secure mobile agent code fully protected from malicious agent platform to provide the integrity of Mobile Agent System.

Keywords: Mobile Agents, Aglet, Homomorphic Encryption, ElGamal Algorithm, Malicious Hosts.

1 Introduction

Using agents to develop services like electronic commerce over the Internet is thought to be a promising approach for future forms of Internet utilization. Here, several services embodied by independently developed agents are connected with each other to render large scale services. In this new paradigm of service development, the Internet is a shared environment for agents to run on. To make use of this environment efficiently, agents should be able to migrate dynamically among several hosts so that they can directly access the necessary resources, get together on a common host to directly communicate with each other, and perform heavy-load tasks on hosts that have enough room. These agents are called mobile agents [1].

A mobile agent system is a new technology allows computers to communicate and facilitates the mobile computation. This technology is a sub area of distributed system. It comes as alternative method of the Remote Procedure Call (RPC) [2]. A mobile agent is an entity that has ability to travel from host to another based on the mobile agents system’s rules. By using its itinerary table, the mobile agent visits hosts to request services. This technology faces a challenge in the security area and many security aspects should be considered when a mobile agent system is designed. The mobile agents system should protect hosts against malicious hosts and protect the mobile agents against malicious hosts [2].

By using homomorphic encryption, the mobile code cannot be effectively protected against the executing system because the host has full control over its execution and it may potentially fully understand the code and eventually can change it in any way it wants. But the argument is that we can obtain an approach where a host can execute an encrypted function without having to decrypt it. Thus, functions would be encrypted such that the resulting transformation can be implemented as a mobile program that will be executed on a remote host. The executing computer will see the program's clear text instructions but will not be able to understand the function that the program implements.

The benefits from utilizing the mobile agents in various business areas are great. However, this technology brings some serious security risks: one of the most important is the possibility of tampering an agent. In the mobile agent systems the agent’s code and internal data migrate between hosts and could be easy changed during the transmission or at the malicious host site[3].
In this paper, we apply a security approach for a crypto computing technique to the security problem of malicious hosts attacking mobile agents even they are honest or malicious. Although the protection of our approach is currently restricted to some parts of the data, it effectively uses a crypto computing technique which encrypt parts of data and perform some operations on this encrypted data without decrypting it to solve such important security problem. The protocol is based on the idea of encrypting a part of the data being transferred among hosts. Malicious hosts cannot see the encrypted data. They only execute their operations without decrypting the data and return these data to the owner host in order to execute the full operation. Homomorphic encryption is used to make the encrypting function of the code by using ElGamal Encryption to encrypt the protected parts of data. The use of this mechanism foreces the mobile agents application to execute the agent safely until the end.

The rest of the paper is organized as follows: Section 2 describes the main published approaches to protect mobile agents. Section 3, 4, 5 and 6 details the proposed protocol to protect agents from malicious hosts and finally, some conclusions can be found in Section 7.

2 Related Work

Since the beginning of mobile agent research, many security issues have been identified. These issues were classified according to the source of the attack and the entity being attacked: agents against agents, agents against hosts, and hosts against agents. The second category - agents against host platforms - includes threats in which agents perform some malicious action on a resource they can access (e.g. deleting a file). Several mechanisms have been proposed to address the malicious host server problem.

Tan H.K. and Morean L in [4] used obfuscation code that generating executable agents which cannot be attacked by reading or manipulating their code. Their technique is based on transforming the agent code in such a way that it is functionally identical to the original one, but it is impossible to understand it. The approach also establishes a time interval during which the agent and its sensitive data are valid. After this time elapses, any attempt to attack the agent becomes worthless. The major drawback of these techniques is the difficulty in establishing the time required by an attacker to understand an obfuscated code.

In [5], Vigna introduces the idea of cryptographic traces. The running agent takes traces of instructions that alter the agent’s state due to external variables. The host sends a hash of the traces with the results because the complete traces are too large. If the agent owner wants to verify execution, it asks for the traces and executes the agent again. If new execution does not agree with the traces, the host is cheating. The approach not only detects attacks, but it also proves the malicious behavior of the host. This approach has various drawbacks: (1) Verification is only performed in case of suspicion; (2) Each host must store the traces for an indefinite period of time because the origin host can ask for them; and (3) a third trusted party is needed in order to punish malicious behaviors. In [6], Roth presents the idea of mutual protection. In an open environment like the Internet it can be assumed that trustworthy relationships are limited, so collusion between hosts is difficult. For this reason, the agent’s results are saved in a cooperative agent that has a disjoint itinerary. This approach presents two drawbacks: (1) The loss of the cooperative agent implies the loss of the results; (2) The possibility of collusion does not disappear.

In the next section, we discuss about our implementation of Homomorphic encryption.

3 Dynamic Programming

Homomorphic Encryption

In multi-agent systems, multiple autonomous agents sometimes need to solve a combinatorial optimization problem by using their private information. For example, in a combinatorial auction where multiple goods are auctioned simultaneously, agents need to find a combination of bids for disjoint set of goods, so that the sum of the bidding prices is maximized for all goods. The problem is called the winner determination problem and has recently become a very active research field [7, 8, 9].

If there exists a fully trusted agent, the participants can trust the auctioneer, it is possible to gather all private information relevant to the combinatorial optimization problem at this trusted agent; thus this agent can solve the problem using any available centralized optimization technique.

However, we cannot take it for granted that there exists such a trusted agent. For example, in a standard first-price sealed-bid auction [10], where the highest bidder wins and pays his/her own price, the auctioneer might collude with a particular participant and reveal information about incoming bids to that participant during the auction.

The proposed solution to this problem is the secure dynamic programming protocol [11] that utilizes indistinguishable, homomorphic and randomizable public key encryption scheme.
An example application of this protocol is the combinatorial auction, where multiple servers can solve a winner determination problem, i.e., they can find the combination of bids so that the sum of the bidding prices is maximized. Although the servers can compute the optimal solution correctly, the information of the bids that are not part of the optimal solution is kept secret even from the servers [11]. Dynamic programming is a powerful method that can be applied to various combinatorial optimization problems. Dynamic programming [10] was developed by R. Bellman during the late 1950’s. The Secure dynamic programming [11] protocol is described in the following paragraphs based on the problem of finding the longest path in the one-dimensional directed graph shown in Figure 1. This problem is similar to the winner determination problem described in the previous paragraphs.

*Figure 1: One dimensional directed graph*

The graph consists of nodes 0, 1, 2, ..., m with directed links among them. A link is represented as (j, k) where j < k. For each link (j, k), the weight of the link w(j, k) is defined. The goal is to find the longest path from initial node 0 to terminal node m, i.e., to find a path from 0 to m so that the sum of the weights of links are maximized. For simplicity, we assume for each node j (where 0 ≤ j < m), there exists at least one link that starts from j, i.e., there is no dead-end node except m.

We can obtain the length of the longest path from 0 to m by solving the following recurrence formula from node m - 1 to 0:

\[
f(j) = \max_{i} (f(i) + w(j, i))
\]

In this formula, f(j) represents the length of the longest path from j to m which is called the evaluation value of node j. For terminal node m, f(m) is defined as 0. For initial node 0, f(0) represents the optimal solution, i.e., the length of the longest path from 0 to m.

The basic idea of the protocol is as follows:

- We assume there is a weight publisher \( P \) for each link (j, k), and an evaluator \( T \) for each node i. In an auction setting, a weight publisher corresponds to a bidder, and an evaluator corresponds to a part of the multiple auction servers.
- These evaluators cooperatively execute dynamic programming. Evaluator \( T \) knows only its evaluation value \( f(i) \) and does not know any weight of any link.
- The protocol is outlined as follows:
  - The weight publisher \( P \) encrypts its weight \( w(j, k) \) using \( T \)’s encryption function.
  - Evaluator \( T \) (who cannot decrypt this information) then calculates the encryption of \( w(j, k) + f(k) \).
  - Evaluator \( T \) then calculates \( f(j) \) by decrypting a part of this encryption without knowing \( w(j, k) \).

To implement this protocol in aglets the encryption scheme we used the ElGamal encryption scheme which is an indistinguishable, homomorphic and randomizable public key encryption scheme.

In the next section, we discuss how ElGamal Encryption scheme works.

### 4 ElGamal Encryption Decryption Algorithm

The ElGamal encryption system parameter consist of a prime number (p) and an integer number (g). Also, a private key (a) is used to generate the public key (y) as given in Equation 1 such that (a) is an integer between 1 and p-1.

\[
Y = g^a \mod p
\]

(1)

The public key for ElGamal encryption consist of the triple (p, g, y). To encrypt a data (m), a random integer number (k) is choosen such that between 1 and p-2. The data must be converted to numbers before producing the ciphertext (c) which consist of the pair (y1, y2) calculated as given by Equation 2 a and b

\[
y1 = g^k \mod p
\]

(2a)

\[
y2 = m^y^k \mod p
\]

(2b)

The decryption process is applied in a reverse order such that the encrypted data (y1,y2) is used with private key (a) and the prime number (p) to retrieve the original data (m) as given by Equation 3.

\[
m = y2/y1^a \mod p
\]

(3)

The main functions of the algorithm can be summarized by the following steps:
Step 1 initialization
1. Choose a random prime number (p)
2. Choose the value of the private key (a)
3. Choose the value for the integer number (g)
4. Compute the value of public key y = g^a mod p
5. Publicize the value (p, g, y) and keep the private key

Step 2 prepare the data
1. Choose the value of random number (k)
2. Get the data to be encrypted
3. Divide the long data into smaller blocks
4. Convert the characters to numbers

Step 3 encryption
1. Encrypt the data by compute the ciphertext (y1, y2)

Step 4 decryption
1. Decrypt the data be decipher the ciphertext (y1, y2)

5 Implementation of Dynamic Programming Homomorphic Encryption

The implementation of the protocol is done for the problem of finding the longest path in the one-dimensional directed graph in Figure 1. This problem and the protocol used to solve the problem are described in detail in the previous section. The details of our implementation can be described as follows.

In our implementation there are two evaluator aglets, evaluator 1 aglet and evaluator 2 aglet, one each for the nodes 1 & 2 respectively. There is also one application aglet. The assumption here is that each one of the evaluators does not try to decrypt the information that it does not need to know to execute the protocol. The flow of our program is given in Figure 2 and it is described in the following steps.

Step 1: The weight publisher (application aglet) for the weight of the link (2, 3) encrypts the weight using ElGamal encryption, which is $e(w(2,3))$. The encrypted weight is then added to $f(3)$ (the evaluation value of node 3). Here, $f(3)$ is 0 since node 3 is the last node. Then, $e(w(2,3)) + f(3)$ is passed as an argument to the evaluator aglet which here is Evaluator 2 aglet and is then dispatched to the destination.

Step 2: The Evaluator 2 aglet at the destination decrypts $e(w(2,3)) + f(3)$ and gets the value of $f(3)$. It then adds this value to $e(w(1, 2))$ which was also passed as an argument from the application aglet. The final value $e(w(1,2)) + f(3)$ is then passed back to the application aglet at the source by message passing.

Steps 3 and 4: The application aglet then passes the following values $e(w(1,2)) + f(3)$, $e(w(1,3))$ and $e(w(0,1))$ as arguments to the Evaluator 1 aglet. Since the Evaluator 1 aglet represents the node 1 and there are two links branching out of node 1, this aglet has to find out the maximum of the two paths, i.e., the maximum of the following two values: $e(w(1,2)) + f(3)$ and $e(w(1,3))$. It then adds the value $e(w(0, 1)$ to the maximum value. This is the final value which is the length of the longest path, and is now passed back to the application aglet by message passing. The Application aglet then decrypts the final result and prints it to the console.

Using the above processes, we can find the maximum of weights, and add a constant to a weight without decrypting it. Since we do not reveal the weights to evaluator 1 and evaluator 2, they have to perform all the operations on the encrypted weights only, and that is made possible by the vector representation of weights.

We can add a constant such as the evaluation value of a node, to an encrypted weight $e(w) = (e_1, ..., e_n)$ without decrypting $e(w)$ nor learning $w$. By shifting and randomizing $e(w)$, we can obtain $e'(w + f) = (E(z), ..., E(z), e'_1, ..., e'_{n-f})$, where $e'$ is a randomization of ciphertext $e$. Due to randomization one can obtain no information about constant $f$ from $e(w)$ and $e'(w + f)$.

Figure 2. The proposed dynamic programming flow
For example, to find the maximum of $e'(w + f)$ and $e(v)$ (encrypted weight of the weight $v$), we first create the product of $e'(w + f)$ and $e(v)$ and then start decrypting the elements in the vector representation of this product. We decrypt the elements from last to the first. Every time after decrypting an element, we check to see if it was a $1$ or $z$. If it was $z$ then the maximum weight is the place number or index number of the element in the vector representation.

6 Testing

For the simplicity of testing, we implement the previous protocol in C++ environment. Application aglet has a passing message property which is not exist in our environment. So, passing encrypted data is done by function calls.

Table 1: The Implementation code

```c++
#include <iostream>
#include <conio.h>
#include <math>

//prototypes
void encrypt(int [], int);
int decrypt(int[]);
int f;
int m;
int result[3];

int main()
{
    int w1 = 2, w2 = 3, w3 = 1;
    int f0 = 6, f1 = 4, f2 = 5, f3 = 0;
    int p = 23;
    int a = 6;
    int g = 11;
    int k = 3;
    int y = pow(g, a);
    y = y % p; // y=9
    int value;

    int fir[8];
    int sec [8];
    int thi [8];

    fir[0] = w1;
    sec[0] = w2;
    thi[0] = w3;


    encrypt(thi, 3);
    cout << "Step1" << endl;
    cout << "Mobile Agent's Application encrypt(\(w(2,3)\)) \);
    cout << "\nThe encrypted weight \(e(\(w(2,3)\))\) is added to \(f(3)\) \);
    cout << "\nThe \(e(\(w(2,3)\)) + f(3)\) is decrypted and passed to Host2\);
    cout << "\n\nStep2\nvalue = decrypt(\(thi\)) \);
    cout << "\nThe Host2 decrypt the \(e(\(w(2,3)\)) + f(3)\) \);
    cout << "\nThen get the value of \(f(3)\) =" << value;
    encrypt(sec, 2);
    cout << "\nThe encrypted weight \(e(\(w(1,2)\))\) \);
    cout << "\nThe \(\(e(\(w(1,2)\)) + f(3)\) is decrypted and passed to Mobile Agent's Application\);
    cout << "\n\nStep3\nencrypt(fir, 1);
    cout << "\nMobile Agent's Application encrypt(\(w(0,1)\)) and pass all weights to Host1\);
    value = decrypt(\(fir\));
    cout << "\nThe value of \(f(2)\) =" << value << " will be to the \(e(\(w(0,1)\))\) and pass it to the Mobile Agent's application\);
    cout << "\n\nStep4\nthe result = " << f2 + w1;

    getch();
    return 0;
}

void encrypt (int arr[], int w)
{
    switch(w)
    {
    case 1 : m = 2;
        f = 5;
        break;
    case 2 : m = 3;
        f = 0;
        break;
    case 3 : m = 1;
        f = 0;
        break;
    }
} 
```
arr[6]=(pow(arr[3],arr[4]));
arr[6]=arr[6]%arr[1];
arr[7]=m*(pow(arr[5],arr[4]));
arr[7]=arr[7]%arr[1];
result[0]=arr[6];
result[1]=arr[7];
result[2]=f;
}

int decrypt( int arr[])
{
    m= arr[7]/(pow(arr[6],arr[2]));
    m=m% arr[1];
    return result[2];
}

The result of this implementation as the following:

The console window show the flow of the program implementation has been described in the previous section.

7 Conclusion
In this paper we have shown the implementation of a security approach to protect the mobile agents against malicious hosts, the security approach that are implemented is Cryptocomputing approach using Homomorphic encryption. In this approach the computation is done on the encrypted data itself without decrypting it which mattern the agent security. The encryption technique used in this approach is ElGamal Encryption Algorithm to encrypt the data partially to provide the integrity for the Mobile Agent System. In this way, we have achieved our goal of protecting agents from being attacked by malicious hosts.

References