Protecting Free Roaming Agents against Result-Truncation Attack

Jianying Zhou
Institute for Infocomm Research
21 Heng Mui Keng Terrace
Singapore 119613
`jyzhou@i2r.a-star.edu.sg`

Jose A. Onieva and Javier Lopez
Computer Science Department
University of Malaga
29071 - Malaga, Spain
{`onieva,jlm`}`@lcc.uma.es`

***Abstract*— Mobile agents are especially useful in electronic commerce, for both wired and wireless environments. Nevertheless, there are still many security issues on mobile agents to be addressed, for example, data confidentiality, non-repudiability, forward privacy, publicly verifiable forward integrity, insertion defense, truncation defense, etc. One of the hardest security problems for free roaming agents is truncation defense where two visited hosts (or one revisited host) can collude to discard the partial results collected between their respective visits. In this paper, we present a new scheme satisfying those security requirements, especially protecting free roaming agents against result-truncation attack.**

Keywords: secure electronic commerce, mobile agent, cryptographic protocol.

I. INTRODUCTION

Mobile agents are software programs that live in computer networks, performing their computations and moving from host to host as necessary to fulfill their goals [2]. Mobile agents are especially useful in electronic commerce, and have attracted lot of research interest. Nevertheless, as stated in [3], there are still many security issues on mobile agents to be addressed. We could classify the security issues of mobile agents as

- protection of the host from malicious code, and
- protection of the agent from a malicious host trying to tamper the code and the agent data.

The community has initially placed more attention in the first problem that is similar to the one existed with Java and ActiveX technologies in which a host has to run software coming from untrusted sources. The most popular solution is *sandbox*, i.e., an agent cannot control the machine in which it is executed.

With respect to the second problem, we can further classify it into two sub-problems. In the first case, a malicious host tries to tamper the agent's code. To address this problem, computing with encrypted functions such as *homomorphic* encryption schemes is under research [4]. In the second case, a malicious host tries to tamper the data carried by the agent. This problem is especially serious for *free roaming* mobile agents that are free to choose their respective next hops dynamically based on the data they acquired from their past journeys. For instance, in a scenario that a free-roaming agent is used to collect offers for an air-ticket, a malicious host may try to "hijack" or "brainwash" the previously collected data to favor its offer. This paper will be focused on the solutions of protecting agent data (or computation results).

The rest of this paper is organized as follows. In Section 2, we outline the security requirements that a free roaming mobile agent should satisfy. In Section 3, we review the previous work on protection of agent data, and point out their weaknesses and limitations. After that, we propose a new scheme in Section 4 that protects the agent data while a mobile agent roams freely in computer networks. We give an informal analysis of our scheme in Section 5, and conclude the paper in Section 6.

II. SECURITY REQUIREMENTS

Suppose a mobile agent departing from host S_0 will obtain a list of encapsulated offers $O_1, \cdots, O_n$ from different hosts $S_1, \cdots, S_n$ that are selected dynamically when the agent roams over the network. The security properties on the agent data protection defined in [2] and extended in [1] are as follows.

- *Data Confidentiality*: Only the originator S_0 can extract the encapsulated offers $O_1, \cdots, O_n$.
- *Non-repudiability*: S_i cannot deny submitting O_i once S_0 receives O_i.
- *Forward Privacy*: No one except the originator S_0 can extract the identity information of the hosts $S_1, \cdots, S_n$ by examining the chain of encapsulated offers.
- *Forward Integrity*: None of the encapsulated offers O_i can be modified.
- *Publicly Verifiable Forward Integrity*: Anyone can check the integrity of the chain of encapsulated offers.
- *Insertion Defense*: No new offer can be inserted in $O_1, \cdots, O_n$ without being detected.
- *Truncation Defense*: No existing offer can be removed from $O_1, \cdots, O_n$ without being detected.

One of the hardest security problems for free roaming agents is truncation defense. In this paper, we present a new scheme satisfying the above security requirements, especially protecting free roaming agents against result-truncation attack.

III. PREVIOUS WORK

Several schemes have been proposed to protect agent data. Yee proposed to use a *Partial Result Authentication Code* (PRAC) to ensure the integrity of the offers acquired from the hosts [5]. In this scheme, an agent and its originator maintain a list of secret keys, or a key generating function. The agent uses a key to encapsulate the collected offer and then destroys the key. However, a malicious host may keep the key or the key generating function. When the agent revisits the host or visits another host conspiring with it, a previous offer or series of offers would be modified, without being detected by the originator.

Karjoth et al. extended Yee's results. In the KAG scheme [2], each host generates a signing key for its successor and certifies the corresponding verification key. Using the received signature/verification key pair, a host signs its partial result and certifies a new verification key for the next host. Their scheme could resist the modification attack in Yee's scheme but not a two-colluder truncation attack. In this attack, two visited hosts (or one revisited host) can collude to discard the partial results collected between their respective visits.

Cheng and Wei further enhanced the KAG scheme to defend the two-colluder truncation attack. In the Cheng-Wei scheme [1], a host is first required to get a counter-signature of its partial result from its predecessor before sending it to the next host. In such a way, any two hosts cannot collude to truncate the agent data collected in the period that the agent visits these two colluding hosts. However, this scheme still suffers from the truncation attack when a special loop is established on the path of a free-roaming agent [6].

IV. OUR PROTOCOL

Here we intend to improve the Cheng-Wei scheme to get rid of its weaknesses. Our new protocol will be effective in defending any two-colluder truncation attack.

Consider a shopping scenario in which an agent departing from host S_0 will obtain a list of offers from different hosts $S_1, \cdots, S_n$ selected dynamically when the agent roams over the network. Among all the security requirements listed in Section 2, we focus our attention on the truncation defense, and in particular, defense against a *two-colluder truncation attack*. In this scenario, an attacker W captures an agent with encapsulated offers $O_1, \cdots, O_{j-1}, O_j, \cdots, O_n$ and colludes with host S_j trying to truncate all the offers after O_j and insert the attacker's offers to get the new chain $O_1, \cdots, O_{j-1}, O'_j, \cdots, O_W$.

A. Assumptions and Notation

A public key infrastructure is assumed in the mobile agent environment. Each host S_i has a certified private/public key pair $(\bar{v}_i, v_i)$. Given a signature expressed as $Sig_{\bar{v}_i}(m)$, we assume that anyone could deduce the identity of S_i from it. The chain of encapsulated offers $O_1, O_2, \cdots, O_n$ is an ordered sequence. Each entry of the chain depends on some of the previous and/or succeeding members. A chaining relation specifies the dependency.

An agent is defined as $A = (I, C, S)$ where I is the identity, C is the code and S is the state of the agent. Both I and C are assumed to be static while S is variable. I is in the form of (ID_A, Seq_A), where ID_A is a fixed identity bit string of the agent and Seq_A is a sequence number which is unique for each agent execution. The originator signs h_A, where $h_A = H(I, C)$ is the agent integrity checksum and $Sig_{\bar{v}_0}(h_A)$ is the *certified agent integrity checksum*. The agent carries this certified checksum, allowing the public to verify the integrity of I and C and deduce the identity of S_0.

Our protocol is similar to the Cheng-Wei scheme and uses a co-signing mechanism in which a host needs the preceding host's signature on its encapsulated offer before sending it to the next host. It also depends on the signatures on the agent integrity checksum generated by the two associated preceding hosts such that the current host is able to verify that the preceding host did not insert two offers in a self-looping mode.

The model and cryptographic notation used in the protocol description is summarized in Tables I and II, respectively.

$S_0 = S_{n+1}$	The originator
$S_i, 1 \leq i \leq n$	A host
$o_i, 1 \leq i \leq n$	An offer from S_i. The identity of S_i is explicitly specified in o_i
$O_i, 1 \leq i \leq n$	An encapsulated offer (cryptographically protected o_i) from S_i
$h_i, 1 \leq i \leq n$	An integrity check value associated with O_i and the next hop
$O_0, O_1, .., O_n$	The chain of encapsulated offers

TABLE I
MODEL NOTATION

r_i	A random number generated by S_i
$(\bar{v}_i, v_i)$	Private and public key pair of S_i
$(\bar{\mu}_i, \mu_i)$	Temporary private and public key pair of S_i
$Enc_{v_i}(m)$	A message m encrypted with the public key v_i of S_i
$Sig_{\bar{v}_i}(m)$	A signature of S_i on message m with its private key $\bar{v}_i$
$Ver(\sigma, v)$	A signature verification function for signature σ with public key v
$H(m)$	A one-way, collision-free hash function
$[m]$	Message m sent via a confidential channel
$A \rightarrow B : m$	A sends message m to B

TABLE II
CRYPTOGRAPHIC NOTATION

B. Protocol Specification

Our protocol consists of three parts: agent creation, agent migration at S_1, and agent migration at S_i $(2 \leq i \leq n)$.

Agent Creation

1. *Offer encapsulation*

$$\begin{array}{ll} S_0: & h_0 = H(r_0, S_1) \\ S_0: & O_0 = Sig_{\bar{v}_0}(Enc_{v_0}(r_0), I, h_0) \\ S_0: & \sigma_0 = Sig_{\bar{v}_0}(h_0) \end{array}$$

The originator S_0 of an agent first generates a random number r_0 and selects the next host S_1 that the agent will visit. Then S_0 calculates an agent integrity checksum h_0 and creates a signature σ_0. S_0 also encapsulates a dummy offer O_0.

2. *Agent transmission*

$$S_0 \rightarrow S_1: \; O_0, \sigma_0$$

When the agent roams from S_0 to S_1, the agent will carry O_0 and σ_0.

Agent Migration at $\mathbf{S_1}$

3. *Agent verification*

$$\begin{array}{ll} S_1: & \text{receive } O_0, \sigma_0 \\ S_1: & Ver(O_0, v_0), \text{and recover } I, h_0 \\ S_1: & Ver(\sigma_0, v_0) \end{array}$$

When the agent arrives, host S_1 will check the data carried by the agent. It verifies S_0's signature O_0 to identify the sender of the agent. It also verifies S_0's signature σ_0 to identify the agent.

4. *Interactive offer encapsulation*

$$\begin{array}{ll} S_1: & h_1 = H(O_0, r_1, S_2) \\ S_1 \rightarrow S_0: & temp_1 = \\ & Enc_{v_0}(Sig_{\bar{v}_1}(o_1, \mu_1, \sigma_0), r_1), h_1, \mu_1 \\ S_0 \rightarrow S_1: & O_1 = Sig_{\bar{v}_0}(temp_1) \\ S_1: & Ver(O_1, v_0) \\ S_1: & \sigma_1 = Sig_{\bar{v}_1}(h_1) \end{array}$$

Host S_1 generates a pair of its temporary private and public keys $(\bar{\mu}_1, \mu_1)$ and a random number r_1. S_1 also selects the next host S_2 that the agent will visit. Then S_1 calculates an agent integrity checksum h_1 and a partial encapsulated offer $Enc_{v_0}(Sig_{\bar{v}_1}(o_1, \mu_1, \sigma_0), r_1)$.

S_1 forms $temp_1$ which also includes its temporary public key μ_1. $temp_1$ is then sent to S_0 for counter-signing. (It is assumed that $temp_1$ is sent over an authenticated channel. S_0 will record the agent departed from it and only sign $temp_1$ once.) O_1 not only represents S_1's encapsulated offer, but also certifies that μ_1 is S_1's temporary public key.

Upon receipt and verification of O_1 from S_0, S_1 finally signs h_1 to get σ_1.

5. *Agent transmission*

$$S_1 \rightarrow S_2: O_0, O_1, \sigma_0, [\sigma_1]$$

When the agent roams from S_1 to S_2, the agent will carry O_0, O_1 and σ_0, σ_1. To provide forward privacy of identities of hosts that the agent has visited (excluding the originator S_0), σ_1 is transmitted over a confidential channel from S_1 to S_2.

Agent Migration at $\mathbf{S_i}$ $(2 \leq i \leq n)$

6. *Agent verification*

$$\begin{array}{ll} S_i: & \text{receive } O_0, \cdots, O_{i-1}, \sigma_{i-2}, \sigma_{i-1} \\ S_i: & Ver(O_0, v_0), \text{and recover } I, h_0 \\ S_i: & Ver(O_1, v_0), \text{and recover } h_1, \mu_1 \\ S_i: & Ver(O_k, \mu_{k-1}), \text{and recover } h_k, \mu_k \\ & \text{recusively for } 2 \leq k \leq i-1 \\ S_i: & Ver(\sigma_{i-2}, v_{i-2}) \\ S_i: & Ver(\sigma_{i-1}, v_{i-1}) \\ S_i: & \text{verify } S_{i-2} \neq S_{i-1} \end{array}$$

As in Step 3, when the agent migrates from S_{i-1} to S_i, S_i will check the data carried by the agent. It recovers $\mu_1, \cdots, \mu_{i-1}$ from $O_1, \cdots, O_{i-1}$, and verifies these encapsulated offers with the corresponding temporary public keys. It also verifies the certified agent checksums $\sigma_{i-2}, \sigma_{i-1}$ and more importantly, make sure two hosts S_{i-2} and S_{i-1} are different. Otherwise, a truncation attack colluding with such a host is possible.

7. *Interactive offer encapsulation*

$$\begin{array}{ll} S_i: & h_i = H(O_{i-1}, r_i, S_{i+1}) \\ S_i \rightarrow S_{i-1}: & temp_i = \\ & Enc_{v_0}(Sig_{\bar{v}_i}(o_i, \mu_i, \sigma_{i-2}, \sigma_{i-1}), r_i), \\ & h_i, \mu_i \\ S_{i-1} \rightarrow S_i: & O_i = Sig_{\bar{\mu}_{i-1}}(temp_i) \\ S_i: & Ver(O_i, \mu_{i-1}) \\ S_i: & \sigma_i = Sig_{\bar{v}_i}(h_i) \end{array}$$

This step is similar to Step 4, but the format of S_i's partial encapsulated offer is slightly different which links to two preceding hosts S_{i-1} and S_{i-2}. In addition, the key used for counter-signing $temp_i$ by S_{i-1} is its temporary key $\bar{\mu}_{i-1}$ instead of $\bar{v}_{i-1}$. In such a way, all encapsulated offers can be verified publicly without revealing the real identities of those counter-signers.

8. *Agent transmission*

$$S_i \rightarrow S_{i+1}: \; \{O_k | 0 \leq k \leq i\}, [\sigma_{i-1}, \sigma_i]$$

This step is similar to Step 5. The agent will carry all the encapsulated offers $O_0, \cdots, O_i$ when it migrates from S_i to S_{i+1}. In addition, both σ_{i-1} and σ_i need to be transmitted over a confidential channel in migration in order to protect the privacy of those identities.

V. SECURITY ANALYSIS

Here we give a brief analysis of our protocol with respect to the security requirements outlined in Section 2.

Data Confidentiality Each offer o_i $(i = 1, \cdots, n)$ that is encapsulated in O_i is encrypted with the originator S_0's public key v_0. Only S_0 can decrypt it to extract the offer, thus confidentiality is preserved.

Non-repudiability Each offer o_i ($i = 1, \cdots, n$) that is encapsulated in O_i is signed by S_i with $\bar{v}_i$. Therefore, S_i cannot deny its offer o_i once the agent carrying O_i returns to the originator S_0.

Forward Privacy Each offer o_i ($i = 1, \cdots, n$) that is encapsulated in O_i is first signed by S_i but then encrypted with S_0's public key v_0. Therefore, the identity of S_i will not be disclosed to others (except S_0) by examining O_i. In addition, as a random number r_i is used in computing the checksum h_i, it reveals no identity information by examining h_i. However, as σ_i will be sent to S_{i+1} and S_{i+2} in order to verify that two adjacent hosts are different on the agent migration path, the identity of S_i will be disclosed to S_{i+1} and S_{i+2}. This implies a slight weakening of forward privacy in our protocol.

Forward Integrity Each offer o_i ($i = 1, \cdots, n$) that is encapsulated in O_i is signed by S_i. Any change to the signed offer will be detected. Furthermore, even S_i cannot change its own encapsulated offer O_i in the chain $O_1, \cdots, O_i, \cdots, O_n$ without being detected. Suppose S_i wants to replace o_i with o'_i. To make this change undetected, S_i needs to get a new counter-signature $O'_i = Sig_{\bar{\mu}_{i-1}}(temp'_i)$ from S_{i-1} which should also satisfy $H(O_i, r_{i+1}, S_{i+2}) = H(O'_i, r_{i+1}, S_{i+2})$. Even if S_{i-1} is willing to collude on generation of O'_i, the equation will not be satisfied under our assumption of collision-free hash function.

Publicly Verifiable Forward Integrity Each encapsulated offer O_i ($i = 1, \cdots, n$) contains S_i's temporary public key μ_i that is certified by S_{i-1} with its temporary private key $\bar{\mu}_{i-1}$. With O_i, anyone can obtain μ_i and use it to verify O_{i+1}. Therefore, the integrity of $O_1, \cdots, O_n$ is publicly verifiable.

Insertion Defense As all encapsulated offers $O_1, \cdots, O_n$ are chained, if a new encapsulated offer O_x is inserted between O_i and O_{i+1} without being detected, some chaining relations have to be changed in O_i and O_{i+1}. Suppose an attacker S_x tries to insert O_x as follows.

$$\begin{aligned}
&h_x = H(O_i, r_x, S_{i+1})\\
&temp_x = Enc_{v_0}(Sig_{\bar{v}_x}(o_x, \mu_x, \sigma_{i-1}, \sigma_i), r_x), h_x, \mu_x\\
&O_x = Sig_{\bar{\mu}_i}(temp_x)\\
&\sigma_x = Sig_{\bar{v}_x}(h_x)
\end{aligned}$$

This implies that S_x at least needs to ask S_i to counter-sign $temp_x$, and ask S_{i+1} to replace its signature in $temp_{i+1}$ as $Sig_{\bar{v}_{i+1}}(o_{i+1}, \mu_{i+1}, \sigma_i, \sigma_x)$. So it is impossible for S_x to insert O_x without collusion with S_i and S_{i+1}.

Truncation Defense The chaining mechanism used in the insertion defense also works for the truncation defense. Suppose an attacker S_x tries to truncate the encapsulated offers $O_1, \cdots, O_i, O_{i+1}, \cdots$ from O_{i+1} thereafter and may also add O_x after O_i. Then, S_x needs to revise O_i as follows.

$$\begin{aligned}
&h'_i = H(O_{i-1}, r_i, S_x)\\
&temp'_i = Enc_{v_0}(Sig_{\bar{v}_i}(o_i, \mu_i, \sigma_{i-2}, \sigma_{i-1}), r_i), h'_i, \mu_i\\
&O'_i = Sig_{\bar{\mu}_{i-1}}(temp'_i)\\
&\sigma'_i = Sig_{\bar{v}_i}(h'_i)
\end{aligned}$$

Obviously, S_x is unable to make the above revisions without collusion with S_i and S_{i-1}. In other words, our protocol defends against truncation attacks if there are no more than two colluders. A straightforward extension of our protocol is possible to defend truncation attacks with more colluders.

Our protocol also resists truncation attacks even if a loop like "$\cdots, S_{i-2}, S_{i-1}, S_i, S_{i+1}, \cdots$ where $S_{i-2} = S_i$" is formed in the roaming path. (This specific attack broke the Cheng-Wei scheme as pointed out in [6].) In this case, S_i and S_{i-2} are the same host. S_{i-2} might substitute a new temporary key pair $(\bar{\mu}''_{i-1}, \mu''_{i-1})$ for S_{i-1} in $temp_{i-1}$ and generate a new O''_{i-1} such that $O''_{i-1} = Sig_{\bar{\mu}_{i-2}}(temp''_{i-1})$, then S_i uses $\bar{\mu}''_{i-1}$ to generate a new O''_i such that $O''_i = Sig_{\bar{\mu}''_{i-1}}(temp''_i)$. However, such a truncation attack by forging S_{i-1}'s temporary key pair will be detected by S_0 when S_0 receives O_{i-1}. S_0 will find that μ''_{i-1} counter-signed by S_{i-2} is different from μ_{i-1} signed by S_{i-1} in $Sig_{\bar{v}_{i-1}}(o_{i-1}, \mu_{i-1}, \sigma_{i-3}, \sigma_{i-2})$.

VI. CONCLUSION

Mobile agents play an important role in electronic commerce, for both wired and wireless environments. A known vulnerability in existing schemes is the *truncation attack* where two hosts visited by a mobile agent can collude to discard the partial results collected by the agent between their respective visits without being detected by the originator of the agent.

In this paper, we proposed a new scheme that is effective in defending against the truncation attack. We also gave a brief analysis to demonstrate that our scheme satisfies other security requirements on the protection of agent data.

ACKNOWLEDGEMENT

The second author's work was done during his attachment to Institute for Infocomm Research under its sponsorship.

REFERENCES

[1] J. Cheng and V. Wei. Defenses against the truncation of computation results of free-roaming agents. In *4th International Conference on Information and Communications Security*, volume LNCS 2513, pages 1–12, December 2002.

[2] G. Karjoth, N. Asokan, and C. Gülcü. Protecting the computation results of free-roaming agents. In *Mobile Agents*, volume LNCS 1477, pages 195–207, September 1998.

[3] G. Karjoth and J. Posegga. Mobile agents and telcos' nightmares. Technical Report 55(7/8):29-41, IBM, 2000.

[4] T. Sander and C. Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, volume LNCS 1419, pages 44–60, 1998.

[5] B. Yee. A sanctuary for mobile agents. In *Secure Internet Programming*, pages 261–273, 1999.

[6] J. Zhou, J. Onieva, and J. Lopez. Analysis of a free roaming agent result-truncation defense scheme. In *2004 IEEE Conference on Electronic Commerce*, July 2004.