Analysis of a Free Roaming Agent Result-Truncation Defense Scheme

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 play an important role in electronic commerce. Security in free-roaming agents is especially hard to achieve when the mobile code is executed in hosts that may behave maliciously. Some schemes have been proposed to protect agent data (or computation results). However, a known vulnerability of these techniques is the truncation attack *where two visited hosts (or one revisited host) can collude to discard the partial results collected between their respective visits. Cheng and Wei proposed a scheme in ICICS'02 to defense against the truncation of computation results of free-roaming agents [1]. Cheng-Wei scheme is effective against such an attack in most cases. However, we demonstrate that it still suffers from the truncation attack when a special loop is established on the path of a free-roaming agent. We further propose two amendments to Cheng-Wei scheme to avoid such an attack.*

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

1. 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]. *Free roaming* mobile agents are free to choose their respective next hops dynamically based on the data they acquired from their past journeys.

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 free-roaming agent security goals 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 and come out with some solutions, since the problem is similar to the one that already existed with Java and ActiveX technologies in which the host has to run software coming from untrusted sources. The most popular solution for such a problem is rather simple, the *sandbox* solution, i.e. an agent cannot control the machine in which it is executed. The agent is executed in a sandbox that blocks the access of the agent to the real machine. This feature has been implemented in Java Mobile Code, Telescript, and SafeTcl.

With respect to the second problem, we can distinguish two principal 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 [5]. In the second case, a malicious host tries to tamper the data carried by the agent. 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).

Suppose an agent departing from host S_0 will obtain a list of encapsulated offers $O_1, \cdots, O_n$ from different hosts $S_1, \cdots, S_n$ 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.

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

- *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.

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 [6]. 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 [2]. In the KAG scheme, 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 proposed a scheme in ICICS'02 to defense against the truncation of computation results of free-roaming agents [1]. Cheng-Wei scheme is effective against such an attack in most cases. However, we demonstrate that it still suffers from the truncation attack when a special loop is established on the path of a free-roaming agent. We further propose two amendments to Cheng-Wei scheme to avoid such an attack.

The rest of this paper is organized as follows. In the next section, we sketch Cheng-Wei Protocol presented in [1]. In Section 3, we demonstrate our attack scenario. In Section 4, we propose two solutions to such an attack. We end the paper with conclusions in Section 5.

2. Cheng-Wei Protocol

We first sketch Cheng-Wei Protocol (N1) presented in [1]. In this protocol, Cheng and Wei considered 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 properties that the protocol claims to achieve is 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 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 important definition given by Cheng and Wei in order to avoid *interleaving* attacks (proposed by Roth [4]) is as follows.

> 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.

In interleaving attacks, the attacker tries to use a previous host as an oracle, running its own agent but with the attacked agent's chain of offers. As the *certified agent integrity checksum* is sent in each transmission between the hosts, each host can verify the consistency of the chain of offers (specially the one belonging to the agent's owner O_0). With this definition, an agent execution is uniquely identified.

Protocol N1 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 description of the original protocol N1 is summarized in Tables 1 and 2, respectively.

Protocol N1 consists of three parts: agent creation, agent migration at S_1, and agent migration at S_i $(2 \leq i \leq n)$ (see Figure 1).

$S_0 = S_{n+1}$	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 1. Model Notation

r_i	A random number generated by S_i
$(\bar{v}_i, v_i)$	Private and public key pair of S_i
$(\bar{\mu}_i, \mu_i)$	Temporally 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 2. Cryptographic Notation

Agent Creation

1. *Offer encapsulation*

$$\begin{array}{ll} S_0: & h_0 = H(r_0, S_1) \\ S_0: & O_0 = Sig_{\bar{v}_0}(Enc_{v_0}(o_0, r_0), I, h_0, \mu_1) \\ S_0: & \sigma_0 = Sig_{\bar{v}_0}(h_0) \end{array}$$

2. *Agent transmission*

$$S_0 \rightarrow S_1 : \ O_0, [\bar{\mu}_1, \sigma_0]$$

Agent Migration at $\mathbf{S_1}$

3. *Agent verification*

$$\begin{array}{ll} S_1: & \text{receive } O_0, \bar{\mu}_1, \sigma_0 \\ S_1: & Ver(O_0, v_0), \text{and recover } I, h_0, \mu_1 \\ S_1: & Ver(\sigma_0, v_0) \end{array}$$

4. *Interactive offer encapsulation*

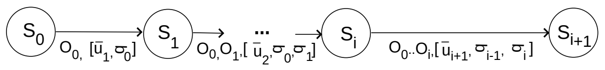

Figure 1. Agent Transmission

$$\begin{array}{ll} S_1: & h_1 = H(O_0, r_1, S_2) \\ S_1 \rightarrow S_0: & temp_1 = Enc_{v_0}(Sig_{\bar{v}_1}(o_1, \sigma_0), r_1), h_1, \mu_2 \\ S_0: & O_1 = Sig_{\bar{v}_0}(temp_1) \\ S_0 \rightarrow S_1: & O_1 \\ S_1: & Ver(O_1, v_0) \\ S_1: & \sigma_1 = Sig_{\bar{v}_1}(h_1) \end{array}$$

5. *Agent transmission*

$$S_1 \rightarrow S_2 : \ O_0, O_1, [\bar{\mu}_2, \sigma_0, \sigma_1]$$

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

6. *Agent verification*

$$\begin{array}{ll} S_i: & \text{receive } O_0, \cdots, O_{i-1}, \bar{\mu}_i, \sigma_{i-2}, \sigma_{i-1} \\ S_i: & Ver(O_0, v_0), \text{and recover } I, h_0, \mu_1 \\ S_i: & Ver(O_1, v_0), \text{and recover } h_1, \mu_2 \\ S_i: & Ver(O_k, \mu_{k-1}), \text{and} \\ & \text{recover } h_k, \mu_{k+1} \text{ recusively for } 2 \leq k \leq i-1 \\ S_i: & Ver(\sigma_{i-2}, v_{i-2}) \\ S_i: & Ver(\sigma_{i-1}, v_{i-1}) \\ S_i: & \text{verify } S_{i-2} \neq S_{i-1} \end{array}$$

7. *Interactive offer encapsulation*

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

8. *Agent transmission*

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

It is assumed that there is an authentication protocol between S_i and S_{i-1} in the co-signing process. S_{i-1} will store the records of the departed agents in such a way that it would only sign once for S_i on the partial results of a particular departed agent.

Each host is responsible for checking and verifying all the messages as well as the chaining relation in the agent state whenever it arrives. If any of the agent verifications fails or S_i receives twice the same agent from S_{i-1} then it rejects the agent.

3. Security Analysis

In this section we present a two-colluder truncation attack on Cheng-Wei Protocol. The protocol bases its security, among other properties, on the impossibility for a host S_j, which tries to collude with a host S_W $(W > j)$, to request a signature on another encapsulated offer O'_j from the previous host S_{j-1} in order to truncate all of the encapsulated offers collected from the hosts between S_{j+1} and S_W.

Our attack scenario is as follows (see Figure 2). A free roaming agent that has visited a host S_j re-visits S_j after visiting another host S_{j+1}, in other words, $S_{j+2} = S_j$. Then, no matter how the agent will roam after visiting S_{j+2}, S_{j+2} can always collude with a host S_W being visited by the agent to truncate all of the encapsulated offers collected after S_{j+2}. S_{j+2} need not collude with S_{j+1} in order to get a new co-signed encapsulated offer O'_{j+2} from S_{j+1}. Instead, S_{j+2} can sign it by itself with the temporary private key $\bar{\mu}_{j+1}$ which was generated by S_j where $S_j = S_{j+2}$!

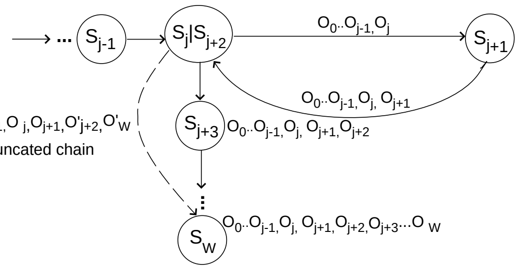
S_{j-1}
S_j|S_{j+2}
O_0..O_{j-1},O_j
S_{j+1}
O_0..O_{j-1},O_j, O_{j+1}
O_0..O_{j-1},O j,O_{j+1},O'_{j+2},O'_W
truncated chain
S_{j+3}
O_0..O_{j-1},O_j, O_{j+1},O_{j+2}
S_W
O_0..O_{j-1},O_j, O_{j+1},O_{j+2},O_{j+3}...O_W

Figure 2. Attack Scenario

Suppose the agent visited $S_0, S_1, \cdots, S_j, S_{j+1}, S_{j+2}\ (= S_j), S_{j+3}, \cdots, S_W$, and collected the encapsulated offers

$$O_0, O_1, \cdots, O_j, O_{j+1}, O_{j+2}, O_{j+3}, \cdots, O_W$$

The processes of interactive offer encapsulation and agent transmission at hosts S_j, S_{j+1} and $S_{j+2}\ (= S_j)$ are as follows.

When the agent is at S_j:

$$\begin{array}{ll}
S_j: & h_j = H(O_{j-1}, r_j, S_{j+1}) \\
S_j \rightarrow S_{j-1}: & temp_j = Enc_{v_0}(Sig_{\bar{v}_j}(o_j, \sigma_{j-2}, \sigma_{j-1}), \\
 & r_j), h_j, \mu_{j+1} \\
S_{j-1}: & O_j = Sig_{\bar{\mu}_{j-1}}(temp_j) \\
S_{j-1} \rightarrow S_j: & O_j \\
S_j: & Ver(O_j, \mu_{j-1}) \\
S_j: & \sigma_j = Sig_{\bar{v}_j}(h_j) \\
S_j \rightarrow S_{j+1}: & \{O_k | 0 \leq k \leq j\}, [\bar{\mu}_{j+1}, \sigma_{j-1}, \sigma_j]
\end{array}$$

When the agent is at S_{j+1}:

$$\begin{array}{ll}
S_{j+1}: & h_{j+1} = H(O_j, r_{j+1}, S_{j+2}) \\
S_{j+1} \rightarrow S_j: & temp_{j+1} = Enc_{v_0}(Sig_{\bar{v}_{j+1}}(o_{j+1}, \\
 & \sigma_{j-1}, \sigma_j), r_{j+1}), h_{j+1}, \mu_{j+2} \\
S_j: & O_{j+1} = Sig_{\bar{\mu}_j}(temp_{j+1}) \\
S_j \rightarrow S_{j+1}: & O_{j+1} \\
S_{j+1}: & Ver(O_{j+1}, \mu_j) \\
S_{j+1}: & \sigma_{j+1} = Sig_{\bar{v}_{j+1}}(h_{j+1}) \\
S_{j+1} \rightarrow S_{j+2}: & \{O_k | 0 \leq k \leq j+1\}, [\bar{\mu}_{j+2}, \sigma_j, \sigma_{j+1}]
\end{array}$$

When the agent is at $S_{j+2}\ (= S_j)$:

$$\begin{array}{ll}
S_{j+2}: & h_{j+2} = H(O_{j+1}, r_{j+2}, S_{j+3}) \\
S_{j+2} \rightarrow S_{j+1}: & temp_{j+2} = Enc_{v_0}(Sig_{\bar{v}_{j+2}}(o_{j+2}, \\
 & \sigma_j, \sigma_{j+1}), r_{j+2}), h_{j+2}, \mu_{j+3} \\
S_{j+1}: & O_{j+2} = Sig_{\bar{\mu}_{j+1}}(temp_{j+2}) \\
S_{j+1} \rightarrow S_{j+2}: & O_{j+2} \\
S_{j+2}: & Ver(O_{j+2}, \mu_{j+1}) \\
S_{j+2}: & \sigma_{j+2} = Sig_{\bar{v}_{j+2}}(h_{j+2}) \\
S_{j+2} \rightarrow S_{j+3}: & \{O_k | 0 \leq k \leq j+2\}, [\bar{\mu}_{j+3}, \sigma_{j+1}, \sigma_{j+2}]
\end{array}$$

Suppose the agent roams to $S_{j+3}, \cdots, S_W$ after visiting S_{j+2}, and then S_W colludes with $S_{j+2}\ (= S_j)$. S_{j+2} can initiate the truncation attack with the following processes of interactive offer encapsulation and agent transmission.

$S_{j+2}\ (= S_j)$ prepares:

$$\begin{array}{ll}
S_{j+2}: & h'_{j+2} = H(O_{j+1}, r_{j+2}, S_W) \\
S_{j+2}: & temp'_{j+2} = Enc_{v_0}(Sig_{\bar{v}_{j+2}}(o'_{j+2}, \sigma_j, \\
 & \sigma_{j+1}), r_{j+2}), h'_{j+2}, \mu_W \\
S_{j+2}: & O'_{j+2} = Sig_{\bar{\mu}_{j+1}}(temp'_{j+2}) \\
S_{j+2}: & \sigma'_{j+2} = Sig_{\bar{v}_{j+2}}(h'_{j+2}) \\
S_{j+2} \rightarrow S_W: & \{O_k | 0 \leq k \leq j+1, O'_{j+2}\}, \\
 & [\bar{\mu}_W, \sigma_{j+1}, \sigma'_{j+2}]
\end{array}$$

When the agent is at S_W:

$$\begin{array}{ll}
S_W: & h'_W = H(O'_{j+2}, r'_W, S_{W+1}) \\
S_W \rightarrow S_{j+2}: & temp'_W = Enc_{v_0}(Sig_{\bar{v}_W}(o'_W, \sigma_{j+1}, \\
 & \sigma'_{j+2}), r'_W), h'_W, \mu_{W+1} \\
S_{j+2}: & O'_W = Sig_{\bar{\mu}_{j+2}}(temp'_W) \\
S_{j+2} \rightarrow S_W: & O'_W \\
S_W: & Ver(O'_W, \mu_{j+2}) \\
S_W: & \sigma'_W = Sig_{\bar{v}_W}(h'_W) \\
S_W \rightarrow S_{W+1}: & \{O_k | 0 \leq k \leq j+1, O'_{j+2}, O'_W\}, \\
 & [\bar{\mu}_{W+1}, \sigma'_{j+2}, \sigma'_W]
\end{array}$$

In the above attack, S_{j+2} and S_W can collude to truncate the encapsulated offers $O_{j+2}, \cdots, O_W$. Then, S_W forwards the truncated chain of encapsulated offers

$$O_0, O_1, \cdots, O_j, O_{j+1}, O'_{j+2}, O'_W$$

to the next host S_{W+1}. As we can see, further checks made by S_{W+1} and even the originator of the agent will not be able to detect such a truncation attack.

4. Amendments

The cause for the attack is that a host may possess sufficient information to forge a signature that is supposed to be generated by its predecessor to establish a publicly verifiable chain of encapsulated offers. To preserve the forward privacy, each encapsulated offer is not signed with a certified private key of a host. Instead, it is signed with a temporary private key generated by a host being just visited. Then, if a loop of $S_i \to S_{i+1} \to S_i$ is formed by a free-roaming agent, the truncation attack will succeed.

A simple solution to avoid this attack is to disallow such a roaming path. In Figure 2, when S_{i+1} is preparing the interactive offer encapsulation, it knows the identity of its predecessor S_i by verifying σ_i and should not select S_i as the next host to be visited.

We could have a more generic solution to avoid the truncation attack even if a loop of $S_i \to S_{i+1} \to S_i$ is formed by a free-roaming agent. To protect against the impersonation of S_{i+1} by S_i in generating an encapsulated offer with a temporary private key $\bar{\mu}_{i+1}$, the key pair $(\bar{\mu}_{i+1}, \mu_{i+1})$ should be generated by S_{i+1} itself rather than by S_i. Then S_{i+1} sends μ_{i+1} to S_i to testify that μ_{i+1} is S_{i+1}'s temporary public key. As $\bar{\mu}_{i+1}$ is only known to S_{i+1}, impersonation is prevented while the forward privacy property of the protocol is preserved. The protocol is modified as follows (see Figure 3.

Agent Creation

1. *Offer encapsulation*

$$\begin{array}{ll} S_0: & h_0 = H(r_0, S_1) \\ S_0: & O_0 = Sig_{\bar{v}_0}(Enc_{v_0}(o_0, r_0), I, h_0) \\ S_0: & \sigma_0 = Sig_{\bar{v}_0}(h_0) \end{array}$$

2. *Agent transmission*

$$S_0 \to S_1: \; O_0, [\sigma_0]$$

Agent Migration at $\mathbf{S_1}$

3. *Agent verification*

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

4. *Interactive offer encapsulation*

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

5. *Agent transmission*

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

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

6. *Agent verification*

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

7. *Interactive offer encapsulation*

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

8. *Agent transmission*

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

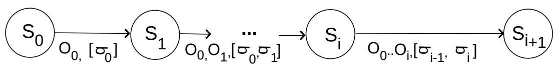

Figure 3. Agent Transmission

With the above change, even if the agent revisits one host, the host still has to request to the previous host for a signature over its new encapsulated offer, and the co-signing process cannot be forged since each host's temporary private key is known to that host only.

5. Conclusion

In mobile code systems more attention is needed in the design of protocols to fulfill those properties. As a consequence of dynamically selected hosts on the path of a free roaming agent, a host can be revisited, and hence, we must ensure that the information it possesses is not enough to carry out a truncation attack by colluding with another party.

In this paper, we briefly reviewed Cheng-Wei scheme for the defense of truncation attack. Their scheme is effective against such an attack in most cases. However, we found that it still suffers from the truncation attack when a special loop is established on the path of a free-roaming agent. We further proposed two solutions to avoid the attack.

References

[1] J. Cheng and V. Wei. Defenses against the truncation of computation results of free-roaming agents. In *Fourth 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] V. Roth. Programming satan's agents. In *1st International Workshop on Secure Mobile Multi-Agent Systems*, May 2001.

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

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