An Improved Free-Roaming Mobile Agent Security Protocol against Colluded Truncation Attacks

Darren Xu
YRC Worldwide Technologies
Overland Park, Kansas, USA
yongnan.xu@yrcw.com

Lein Harn and Mayur Narasimhan
University of Missouri–Kansas City
Kansas City, Missouri, USA
harnl@umkc.edu

Junzhou Luo
Southeast University
Nanjing, P. R. China
jluo@seu.edu.cn

Abstract

This paper proposes an improved free-roaming mobile agent security protocol. The scheme uses "one hop backwards and two hops forwards" chain relation as the protocol core to implement the generally accepted mobile agent security properties. This scheme defends most known attacks, especially colluded truncation attacks and several special cases.

1. Introduction

Mobile agents are software programs which migrate from originating hosts to intermediate servers to generate and collect data, and return to the originators to submit results after completing scheduled tasks. Free-roaming agents are those mobile agents which have no pre-defined migration paths. They select their next hop at each hop they visit based on initial requirements and current conditions. Mobile agents have evoked strong interest due to features such as flexibility, autonomy and efficiency. Applications such as electronic commerce, mobile computing and network management can benefit from the mobile agent technology. However, mobile agents must have strong security properties to protect themselves and the collected data while leaving their homes and migrating to other potentially malicious servers. A malicious server may expose, modify, insert, or truncate data the agent collected from other previously visited servers to benefit itself. One of the diligent attacks is two-colluder truncation attack in which two attackers collude to delete a part of the results collected by an agent from previously visited servers. A general approach to protect mobile agent security is utilizing cryptographic mechanisms to build robust security protocols needed to meet security requirements.

2. Related work

Yee [1] proposed the Partial Result Authentication Codes (PRACs) to protect mobile agent results. In the PRAC schemes, an agent carries key generation functions, or lists of encryption keys or public signature keys, one for each server to be visited. The key generation functions, or the encryption keys or public signature keys are applied to data generated at each host the agent visits. These PRACs ensure data integrity. However, agents must determine how many keys they need to carry before leaving the originators. The agents need to carefully protect the carried keys during their journey, and erase the used keys once they complete their actions on each server. The above requirements are however nearly impossible for free-roaming agents in real network environments.

Karjoth et al. [2] extended Yee's schemes to a set of enhanced security protocols. The KAG protocol family uses digital signatures and hash functions to protect a chain relation. A chain relation links a result from the currently visited host to a result generated at the previous server and the identity of the next hop. By using different combinations of cryptographic mechanisms, each scheme of the protocol family provides different security properties. However, none of the protocols can defend two-colluder truncation attacks.

Karnik et al. [3] introduced the Append Only Container scheme. It is a compact case of the KAG protocol family. The protocol uses an encrypted checksum to build a backward chain relation to link an agent's previous result with the agent's data generated at the currently visited host. The backward chain relation guarantees that only new data can be added to the results the agent collected and no data can be deleted from them. This scheme cannot defend two-colluder truncation attacks.

Corradi et al. [4] integrated the Multiple-Hops Protocol in their mobile security project. Similar to the KAG protocols, this protocol uses a chain relation which includes both backward and forward chaining. At each host, the protocol runs a hash function to compute a cryptographic proof of a result from the previous host, a result generated at the current host, and the identity of the next hop. Like other protocols, this protocol cannot defend two-colluder truncation attacks.

Cheng et al. [5] enhanced the KAG protocols with a co-signing mechanism to defend the two-colluder truncation attacks. In this protocol, a preceding host co-signs a result generated at the current host. Attackers need their preceding non-attackers to co-sign fake offers when they

Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)
0-7695-2655-1/06 $20.00 © 2006 **IEEE**

launch two-colluder truncation attacks, and then their actions can be detected. Like KAG protocols, to reach the so called publicly verifiable forward integrity propriety, this protocol generates a pair of one time secret private and public keys at each host for its successor to use. As Yao et al. [6] and Songsiri [7] point out, the security assurance relies on the assumption that the predecessor does not leak the secret key used by its successor. This requirement to potentially malicious hosts is not realistic. To defend a stemming attack, a special case of two-colluder truncation attacks, the protocol needs to be modified and requires a two-way authentication.

Zhou et al. [8][9] improved Cheng et al.'s protocol. In the protocol, the one time signature key pair is generated by each host rather than the preceding host. The protocol also defends the truncation attack with a special loop. Like Cheng et al.'s protocol, a major issue with the protocol is that a host requires its preceding host to co-sign encrypted data. In real business applications, signing documents assumes legal responsibilities. Since the co-signers cannot check the encrypted data, malicious hosts may ask their preceding hosts to sign encrypted documents and use these signatures against the co-signers later. This protocol cannot defend multiple-colluder truncation attacks. This protocol requires a confidential channel for adjacent hosts to exchange secret data.

Our new protocol addresses all the issues found in the previously discussed protocols, especially solutions to defend the two-colluder attack. The rest of this paper is organized as follows. In Section 3, we give the commonly accepted mobile agent security properties, notations and assumptions used in protocol description. In Section 4, we describe our protocol in detail. In Section 5.1, we analyze the general security properties of the new protocol. In Section 5.2, we discuss two-colluder truncation attacks and other special cases. In Section 6, we conclude the highlights of the protocol.

3. Notations and security properties

To compare with other mobile agent security protocols, we use the similar notions used in other schemes [2][5][9].

Table 1. Model notations

S_0=S_{n+1}	Originator.
S_i, $1 \le i \le n$	Hosts.
o_0	Token from S_0 to identify the agent instance on return.
o_i, $1 \le i \le n$	Offer from S_i. The identity of S_i is explicitly specified in o_i.
O_i, $1 \le i \le n$	Encapsulated offer (cryptographically protected o_i) from S_i.
h_i, $1 \le i \le n$	Integrity check value associated with O_i.
$O_0, O_1, \ldots, O_n$	Chain of encapsulated offers from S_0, $S_1, \ldots S_n$.

Table 2. Cryptographic notations

r_i	Random number generated by S_i.
(Pr_i, Pb_i)	Private and public key pair of S_i.
(tPr_i, tPb_i)	Temporary private and public key pair of S_i.
$Enc_{Pbi}(m)$	Message m encrypted with the public key Pb_i of S_i.
$Sig_{Pri}(m)$	Signature of S_i on message m with its private key Pr_i.
$H(m)$	One-way, collision-free hash function.
A→B: m	A sends message m to B.

A free-roaming agent starts at its originator S_0, and visits other intermediate servers S_i to generate and collect data, and returns to S_0 after completing its itinerary S_0, $S_1, \ldots, S_i, \ldots, S_m, \ldots, S_0$. While the agent migrates to server S_i, S_i encapsulates an offer o_i with other related data to generate its encapsulated offer O_i, and then appends the encapsulated offer to the partial results carried by the agent from the preceding servers. The agent completes its trip and returns to S_0, and S_0 extracts and verifies the encapsulated offers from each visited servers.

Assume the agent has a chain of encapsulated offers O_0, $O_1, \ldots, O_i, \ldots, O_m$ when migrating to S_{m+1}, and S_{m+1} colludes with some of previously visited servers, excluding S_m to attack the offers. Karjoth et al. [2] defined and Cheng, et al. [5] extended following mobile agent security properties based on the assumptions:

- *Data Confidentiality*: Only the originator can extract the offer o_i from the encapsulated offer O_i.
- *Non-repudiability*: Server S_i cannot repudiate its offer o_i once it has been received by the originator S_0.
- *Forward Privacy*: None of the identities of the creators of offer o_i can be extracted by anyone except the originator S_0.
- *Strong Forward Integrity*: None of the encapsulated offers O_k, where $k<m$, can be modified.
- *Public Verifiable Forward Integrity*: Any one can verify the offer o_i by checking whether the chain is valid at O_i.
- *Insertion Resilience*: No offer can be inserted at i unless explicitly allowed; i.e., S_{m+1}. It is impossible for S_{m+1} to insert more than one offer unless S_{m+1} collude with some special L hosts.
- *Truncation Resilience*: Truncation at i is not possible unless some specific L hosts collude with S_i to carry out the attack.

4. The protocol

Our protocol builds a chain relation to link the current encapsulated offer backwards to the previous encapsulated offer and forwards to the identities of next two hops. The protocol is designed to defend all known attacks, especially two-colluder truncation attacks and their special cases, and fix some weaknesses found in other protocols. We describe the protocol as following.

4.1. Agent creation at S_0

The originator S_0 starts the agent with a dummy offer o_0 and a random number r_0. The originator signs the offer and then encrypts the offer and random number by using the originator's secret key to produce a cryptographically protected encapsulated offer $ProtectedO_0=Enc_{Pb0}(Sig_{Pr0}(o_0), r_0)$. The agent selects and then migrates to its next hop S_1 with $ProtectedO_0$.

S_0: Compute $ProtectedO_0=Enc_{Pb0}(Sig_{pr0}(o_0), r_0)$
Decide next hop S_1
$S_0 \rightarrow S_1$: $ProtectedO_0$

4.2. Agent at S_1

When the agent migrates to S_1, it carries the protected encapsulated offer $ProtectedO_0$, instead of the final encapsulated offer O_0. S_1 generates offer o_1 and a random number r_1. S_1 signs its own offer, and then encodes the offer and the random number by using the originator's public key to produce S_1's protected encapsulated offer $ProtectedO_1$. The server S_1 also generates a pair of temporary digital signature keys [tPr_1, tPb_1] for signing its own final encapsulated offer later. The agent then selects its next hop S_2.

S_1: Receive $ProtectedO_0$ from S_0
Compute $ProtectedO_1=Enc_{Pb0}(Sig_{Pr1}(o_1), r_1)$
Generate [tPr_1, tPb_1]
Select S_2

S_1 sends the identity of its next hop S_2 and temporary public key tPb_1 to S_0. To prevent a host from inserting two offers in a self-looping mode [5], S_0 checks and refuses to return its final encapsulated offer O_0 if S_1 and S_2 are the same hosts. Now S_0 is able to build a chain relation h_0 of its previous offer and the next two hops. Since S_0 has no previous offer, the protected encapsulated offer $ProtectedO_0$ is used here. S_0 then signs its final encapsulated offer and forwards it to S_1.

$S_1 \rightarrow S_0$: S_2, tPb_1
$S_0 \rightarrow S_1$: $h_0=H(ProtectedO_0, r_0, S_1, S_2)$, $S_1 \neq S_2$
$O_0=Sig_{Pr0}(ProtectedO_0, h_0, tPb_1)$

After receiving O_0, S_1 can verify O_0 by using S_0's public key and recover $ProtectedO_0$, h_0, and tPb_1, and confirm the $ProtectedO_0$ encapsulated in O_0 is the same as the $ProtectedO_0$ carried with the agent at the time when migrating from S_0 to S_1. This step prevents S_0 from changing its mind to use a different offer o_0 at the time when S_1 asks for the final encapsulated offer O_0. O_0 not only presents S_0's final encapsulated offer, but also certifies that tPb_1 is S_1's temporary digital signature public key. Note not all of calculation or verification steps related to S_0 are necessary in the protocol since S_0 is a trust originator. We keep the actions to generalize the protocol steps.

S_1: $Ver(O_0, Pb_0)$, recover $ProtectedO_1$, h_0, and tPb_1

The agent migrates to S_2 with the final encapsulation offer O_0 from S_0 and the protected encapsulated offer $ProtectedO_1$ from S_1.

$S_1 \rightarrow S_2$: O_0, $ProtectedO_1$

4.3. Agent at S_i

4.3.1. Offer provision. The agent migrates to S_i with all previous encapsulated offers $O_0, O_1, \ldots, O_{i-2}$, plus the protected encapsulated offer $ProtectedO_{i-1}$ from S_{i-1} (instead of the finial encapsulated offer O_{i-1}). After generating a random number r_i and a pair of temporary digital signature keys [tPr_i, tPb_i], S_i computes its protected encapsulated offer $ProtectedO_i$. To prevent reusing of the one time digital signature keys [5], a recording and checking function can be added to the agent. The agent checks and advises S_i to generate a new pair if the same temporary digital signature keys are used before. No confidential information is revealed if the agent only records and checks public key information. As a free-roaming agent, the agent then decides its next hop S_{i+1} based on the initial requirements and the current conditions, such as numbers of visited servers and availability of next hop. The discussions of the checking function and the roaming decision are out of the scope of this paper.

S_i: Receive $O_0, O_1, \ldots, O_{i-2}$, $ProtectedO_{i-1}$ from S_{i-1}
Compute $ProtectedO_i=Enc_{Pb0}(Sig_{Pri}(o_i), r_i)$
Generate [tPr_i, tPb_i]
Select S_{i+1}

4.3.2. Interactive offer encapsulation. S_i informs its previous server S_{i-1} with the next hop identity S_{i+1} and the temporary digital signature public key tPb_i. To prevent potential self-loop attack, S_{i-1} compares S_i with S_{i+1}, and refuses to return its final encapsulated offer O_{i-1} and reports the incident if S_i and S_{i+1} are the same host. S_{i-1} is now able to build a chain relation h_{i-1} of its previous offer O_{i-2} and its next two hops S_i and S_{i+1}. Since the protected encapsulated offer $ProtectedO_{i-1}$ has been computed when the agent was at S_{i-1}, S_{i-1} just simply signs and finalizes the final encapsulation offer O_{i-1} by using its secret key tPr_{i-1}. Since S_i passes the identity of S_{i+1} to S_{i-1} to build the chain relation, this step may reveal S_{i+1}'s identity to S_{i-1}. It is feasible for S_i to pass an encoded identity to fix this weakness but the protocol needs to be modified.

$S_i \rightarrow S_{i-1}$: S_{i+1}, tPb_i
$S_{i-1} \rightarrow S_i$: $h_{i-1}=H(O_{i-2}, r_{i-1}, S_i, S_{i+1})$, $S_i \neq S_{i+1}$
$O_{i-1}=Sig_{tPri-1}(ProtectedO_{i-1}, h_{i-1}, tPb_i)$

4.3.3. Offer verification. Now S_i has all previous offers including the final encapsulated offer O_{i-1} from S_{i-1}. S_i

recovers all of the previous $ProtectedO_k$, h_k, and tPb_{k+1} ($1 \le k \le i\text{-}2$) recursively from O_0, $O_1, \ldots, O_{i-2}$ to verify the offers $O_1, O_2, \ldots, O_{i-1}$ with corresponding public keys. The protocol confirms the $ProtectedO_{i-1}$ encapsulated in O_{i-1} by S_{i-1} in Step 4.3.2 is the same $ProtectedO_{i-1}$ carried over by the agent in Step 4.3.1 to prevent S_{i-1} from changing its mind to use a different offer o_{i-1} after the agent migrates.

S_i: $Ver(O_0, Pb_0)$, recover $ProtectedO_0$, h_0, and tPb_1
$Ver(O_k, tPb_k)$, recover $ProtectedO_k$, h_k, and tPb_{k+1},
$1 \le k \le i\text{-}2$

4.3.4. Agent transmission. S_i forwards all previous offers and its protected encapsulated offer to S_{i+1} if all previous encapsulated offers are verified as valid.

$S_i \rightarrow S_{i+1}$: $O_0, O_1, \ldots, O_{i-1}$, $ProtectedO_i$

4.4. Agent at S_{i+1}

Agent migration at S_{i+1} has the similar processes as at S_i. We outline the protocol at sever S_{i+1} for comparison with steps at S_i.

S_{i+1}: Receive $O_0, O_1, \ldots, O_{i-1}$, $ProtectedO_i$ from S_i
Compute $ProtectedO_{i+1} = Enc_{Pb0}(Sig_{Pri+1}(o_{i+1}), r_{i+1})$
Generate $[tPr_{i+1}, tPb_{i+1}]$
Select S_{i+2}
$S_{i+1} \rightarrow S_i$: S_{i+2}, tPb_{i+1}
$S_i \rightarrow S_{i+1}$: $h_i = H(O_{i-1}, r_i, S_{i+1}, S_{i+2})$, $S_{i+1} \neq S_{i+2}$
$O_i = Sig_{tPri}(ProtectedO_i, h_i, tPb_{i+1})$
S_{i+1}: $Ver(O_0, Pb_0)$, recover $ProtectedO_0$, h_0, and tPb_1
$Ver(O_k, tPb_k)$, recover $ProtectedO_k$, h_k, and tPb_{k+1}, $1 \le k \le i\text{-}1$
$S_{i+1} \rightarrow S_{i+2}$: $O_0, O_1, \ldots, O_i$, $ProtectedO_{i+1}$

4.5. Agent returns to S_0

When the agent returns to the originator S_0, it has all the encapsulated offers O_0, $O_1, \ldots, O_n$. The agent creator S_0 begins to decrypt the offers and extract the data. It uses its public key Pb_0 to recover $ProtectedO_0$ and temporary public key of tPb_1 from O_0, and then uses the temporary public key tPb_1 to recover $ProtectedO_1$ and tPb_2 from the next encapsulated offer. Using these temporary public keys S_0 can extract all the $ProtectedO_i$. The $ProtectedO_i$ can be decrypted using the public key Pb_0 of S_0 and the offers $o_1, \ldots, o_n$ can be obtained.

5. Security analysis

Here we analyze how our protocol achieves the security properties defined in Section 3. We assume the agent's itinerary is $S_0, S_1, \ldots, S_{i-2}, S_{i-1}, S_i \ldots, S_m, \ldots, S_0$, and collected encapsulated offers are $O_0, O_1, \ldots, O_{i-2}, O_{i-1}, O_i, \ldots, O_m, \ldots, O_n$.

5.1. General security properties

5.1.1. Data confidentiality. Each offer o_i is encrypted by S_0's public key Pb_0. Only the originator can decrypt $Enc_{Pb0}(Sig_{Pri}(o_i), r_i)$ and extract o_i.

5.1.2. Non-repudiability. Each offer o_i is signed by S_i as $Sig_{Pri}(o_i)$. S_i cannot deny its offer o_i after S_0 receives the offer and verifies the signature.

5.1.3. Forward privacy. Each offer o_i is signed by server S_i and then encoded by S_0's public key Pb_0 as $Enc_{Pb0}(Sig_{Pri}(o_i), r_i)$. A random number is included in the checksum $h_i = H(O_{i-1}, r_i, S_{i+1}, S_{i+2})$ so no server identity information is exposed by examining h_i. tPb_i is a temporary public key and no identity information is associated with it. With no information revealed from the three components of O_i, no one except S_0 can identify S_i by examining O_i.

5.1.4. Strong forward integrity. Assume an attacker S_m holds encapsulation offers $O_0, O_1, \ldots, O_{i-1}, O_i, \ldots, O_{m-1}$, and modifies or replaces O_{i-1} with O_{i-1}'. O_{i-1} is one of the components in the checksum $h_i = H(O_{i-1}, r_i, S_{i+1}, S_{i+2})$ in the encapsulated offer O_i. Since O_i is intact, the chain relation $h_i = H(O_{i-1}, r_i, S_{i+1}, S_{i+2})$ must be hold true, i.e. $H(O_{i-1}', r_i, S_{i+1}, S_{i+2}) = H(O_{i-1}, r_i, S_{i+1}, S_{i+2})$. This violates the assumption that the hash function H is collision-free. It is impossible for an attacker to modify or replace any offer without changing the next encapsulated offer if a collision-free hash function is used in the protocol.

5.1.5. Publicly verifiable forward integrity. As described in 4.3.3, any server S_i can recover h_0 and tPb_1 from O_0, and h_k and tPb_{k+1} from $O_k = Sig_{tPrk}(ProtectedO_k, h_k, tPb_{k+1})$ ($1 \le k \le i\text{-}2$) recursively to verify the previous encapsulated offers $O_0, O_1, \ldots, O_{i-1}$.

5.1.6. Insertion defense. Assume an attacker S_m inserts an offer O_x between O_{i-1} and O_i to change the encapsulation offers to $O_0, \ldots, O_{i-1}, O_x, O_i, \ldots, O_{m-1}$. Since any two of the ordered encapsulated offers has a chain relation between them, the inserted O_x is a fake O_{i-1} to O_i. As discussed in 5.1.4, O_x cannot be inserted because the chain relation $h_i = H(O_{i-1}, r_i, S_{i+1}, S_{i+2})$ cannot be hold true after the insertion.

5.1.7. Truncation defense. Assume an attacker S_m truncates all encapsulated offers after O_{i+1}, and then appends its own offer O_m. The new chain of encapsulated offers is now $O_0, O_1, \ldots, O_i, O_{i+1}, O_m$. Since O_i is intact, the chain relation $h_i = H(O_{i-1}, r_i, S_{i+1}, S_{i+2})$ must be hold true, i.e. $H(O_{i-1}, r_i, S_{i+1}, S_{i+2}) = H(O_{i-1}, r_i, S_{i+1}, S_m)$. This violates the assumption that the hash function H is collision-free.

5.2. Colluded truncation attacks

Some of other mobile agent security protocols use chain relations to defend truncation attacks. As Karjoth et al. pointed out [2], inclusion of an identity of the next hop in a chain relation guarantees that no one else except the chained next host can append the next offer. In other chain relation based schemes, including Karjoth et al.'s protocols, S_{i-1} retains the chain relation with its previous encapsulated offer O_{i-2} and next one hop S_i. Since only one forward hop is included, the chain relation h_{i-1} at S_{i-1} only guarantees that S_i can append the next offer but cannot prevent S_i from modifying the next hop identity in its own chain relation h_i when S_i joins a colluded truncation attack at a later time. Since S_i can modify its next hop in its own chain relation, S_i is able to collude with S_m to truncate the offers between them and append new offers without being detected. This is why the one-hop forward chain relation based schemes cannot defend colluded truncation attacks without other protection mechanisms.

In our protocol, S_{i-1} builds the chain relation $h_{i-1}=H\,(O_{i-2}, r_{i-1}, S_i, S_{i+1})$ with next two hops S_i and S_{i+1}. The inclusion of S_i and S_{i+1} guarantees that no one else except hosts S_i and S_{i+1} can append the next two offers O_i and O_{i+1}. If O_{i-1} is intact, truncation against O_i and/or O_{i+1} will break the chain relation. The chain relation h_{i-1} may not be able to prevent S_i and S_{i+1} from changing their own chain relations, but it guarantees that only O_i from S_i and O_{i+1} from S_{i+1} can follow O_{i-1}. This property can be used to defend two-colluder truncation attack and many of its special cases. We discuss different scenarios as following.

5.2.1. Two-colluder truncation attack. Assume an agent migrates from a colluder S_i to a non-colluder S_{i+1} and some other non-colluder hosts, then arrives at another colluder host S_m. S_m and S_i leave O_{i-1} intact and collude to truncate O_i and/or afterward. Since O_{i-1} is intact, only S_i and S_{i+1} can append O_i and O_{i+1} after O_{i-1}. If O_i and/or O_{i+1} are truncated, the host identities of the new offers after O_{i-1} cannot satisfy the chain relation $h_{i-1}=H(O_{i-2}, r_{i-1}, S_i, S_{i+1})$ without violating the collusion-free hash function assumption. So the truncation against O_i and O_{i+1} cannot happen or the action will be detected. Since S_{i+1} is not an attacker and O_{i+1} is intact, the chain relation $h_{i+1}=H(O_i, r_{i+1}, S_{i+2}, S_{i+3})$ not only prevents truncations against O_{i+2} and/or O_{i+3}, but also prevents S_i from changing O_i based on the discussion in 5.1.4. Recursively, we can prove that no offers can be truncated or modified with this attack.

5.2.2. Growing a fake stem attack. The combined attack where an attacker simultaneously truncates offers and appends fake offers is referred as *growing a fake stem attack* [2]. In one-hop chain relation based schemes, if O_{i-1} is intact, S_m cannot truncate O_i but can truncate offers after O_i, and collude with S_i to change S_i's chain relation h_i to add new offers. In our protocol, if O_{i-1} is intact, based on the discussion in 5.2.1, the chain relation h_{i-1} prevents S_i and S_m from truncating offers and adding new offers between them so the *growing a fake stem attack* cannot happen.

5.2.3. Multiple-colluder truncation attack. It is possible that multiple (three or more) colluders exist. Assume S_m holds partial encapsulated offers from $S_0,\ldots,S_{i-1}, S_i, S_{i+1},\ldots, S_x, S_{x+1},\ldots, S_{m-1}$, and S_i, S_x, and S_m ($i<x<m$) leave O_{i-1} intact and collude to truncate O_x and/or afterwards. In our protocol, S_{i-1} builds the chain relation with next two hops S_i and S_{i+1}. Only if $x=i+1$, ie, where S_i and S_x are adjacent, S_i and S_x can change their own chain relations, and collude with the third attacker S_m to truncate offers between S_x and S_m. Otherwise, based on the discussion in 5.2.1, the attack cannot be successful. In other words, our protocol defends multiple colluder truncation attacks as long as any two of the colluders are not adjacent. This protocol can be extended to overcome the limitation, but the protocol process will be too complex to implement.

5.2.4. Revisiting attack. In *revisiting attack* [8], an agent leaves S_i, visits some other hosts, S_{i+1}, for example, and then revisits $S_i(S_i=S_{i+2})$. $S_i(S_{i+2})$ then colludes with S_m to truncate O_{i+2} and/or afterwards. In this case, the visited servers are $S_0,\ldots, S_{i-1}, S_i, S_{i+1}, S_{i+2}(S_i), S_{i+3},\ldots, S_m$, and the encapsulated offers before the attack are $O_0,\ldots, O_{i-1}, O_i, O_{i+1}, O_{i+2}, O_{i+3},\ldots, O_{m-1}$. In our protocol, similar to the discussion in 5.2.1, since O_{i+1} is intact, the chain relation $h_{i+1}=H(O_i, r_{i+1}, S_{i+2}, S_{i+3})$ assures that O_{i+2} and O_{i+3} cannot be truncated. Using the same argument recursively, we can prove no other offers can be truncated as well.

5.2.5. Interleaving attack. In *interleaving attack*, attackers may truncate encapsulated offers and replace the agent to insert fake offers [10]. Assume the visited servers are $S_0,\ldots, S_i, S_{i+1}, S_{i+2},\ldots, S_m$, and the encapsulated offers before the attack are $O_0,\ldots, O_i, O_{i+1}, O_{i+2},\ldots, O_{m-1}$. S_m truncates the encapsulated offers to $O_0,\ldots, O_{i-1}, O_i$, replaces the agent with its own version, and colludes with S_i to change the chain relation in O_i, and migrates the new agent to next hop. Since the new agent is created by S_m, S_m can select an itinerary $S_{i+1}', S_{i+2}'\ldots$, and make the agent back to S_m. If this happens, S_m gets a new encapsulated offer chain $O_0,\ldots, O_i, O_{i+1}', O_{i+2}',\ldots, O_{m-1}'$. S_m switches back to the original agent, appends its own encapsulated O_m and continues the original itinerary. In other protocols [5][9], a certified agent integrity checksum is used to allow the public to verify the agent and reject it if the agent is not from S_0. In our protocol, since O_{i-1} is intact, only S_i and S_{i+1} can append O_i and O_{i+1} after O_{i-1}. If S_m creates a new agent, the new agent must be sent to S_{i+1} again to keep the chain relation $h_{i-1}=H(O_{i-2}, r_{i-1}, S_i, S_{i+1})$ valid. S_{i+1} is able to check the encapsulated offer chain $O_0,\ldots, O_i$, and rejects this migration if this is a repeated action.

6. Conclusion

Our scheme uses a "one hop backwards and two hop forwards" chain relation to build a free-roaming agent security protocol to implement all of the generally accepted security properties. The protocol is designed especially to defend the two-colluder truncation attack and many of its special cases, including growing a fake stem attack, revisiting attack and interleaving attack. This protocol defends multiple colluder truncation attacks in most cases. Compared with other free-roaming agent security schemes, this scheme has relatively simple protocol processes and minimum requirements on network and security environments, such as no requirements for confidential channels and no co-signs on encrypted contents. Like other schemes with ability of defending colluder truncation attacks [5][8][9], this scheme requires communications between servers after agent migrations and may increase communication overhead or cause the process fail if a previous server is not available. This protocol offers many unique and attractive features to protect free-roaming agents in a distributed environment.

Acknowledgement

This work is supported by Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No. BM2003201.

References

[1] B.S. Yee. “A sanctuary for mobile agents”. *Technical Report CS97-537*, UC San Diego, Department of Computer Science and Engineering, April 1997.

[2] G. Karjoth, N. Asokan, and C. Gülcü. “Protecting the computation results of freeroaming agents”. In *Proc. Second International Workshop on Mobile Agents (MA ’98), K. Rothermel and F. Hohl, editors, LNCS 1477*, pp. 195 - 207, Springer-Verlag, 1998.

[3] N. M. Karnik and A. R. Tripathi. “Security in the Ajanta Mobile Agent System”. *Technical Report TR-5-99*, University of Minnesota, Minneapolis, MN 55455, U. S. A., May 1999.

[4] A. Corradi, R. Montanari, and C. Stefanelli. “Mobile agents Protection in the Internet Environment”. In *The 23rd Annual International Computer Software and Applications Conference (COMPSAC ’99)*, pages pp. 80–85, 1999.

[5] 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.

[6] M. Yao, E. Foo, E. P. Dawson and K. Peng. “An Improved Forward Integrity Protocol for Mobile Agents”. In *proceedings of 4th International Workshop on Information Security Applications (WISA 2003)*, volume 2908 of Lecture Notes in Computer Science, pages 272--285. Springer-Verlag, 2004. ISBN: 3-540-20827-5.

[7] Suphithat Songsiri. "A New Approach for Computation Result Protection in the Mobile Agent Paradigm". In *Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC 2005)*, 27-30 June 2005, Murcia, Cartagena, Spain.

[8] J. Zhou, J. Onieva, and J. Lopez. "*Analysis of a Free Roaming Agent Result-Truncation DefenseScheme*". In *Proceedings of 2004 IEEE Conference on Electronic Commerce*, pages 221--226, San Diego, USA, July 2004, IEEE Computer Society Press.

[9] J. Zhou, J. Onieva, and J. Lopez. "Protecting Free Roaming Agents against Result-Truncation Attack". In *Proceedings of 60th IEEE Vehicular Technology Conference*, pages 3271--3274, Los Angles, USA, September 2004, IEEE Vehicular Technology Society Press.

[10] V. Roth. “On the robustness of some cryptographic protocols for mobile agent protection.” In *Proceedings of the 5th International Conference on Mobile Agents (MA 2001)*, volume 2240 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 2001.