View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Institutional Knowledge at Singapore Management University

Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems | School of Information Systems

10-2009

Secure Mobile Agents with Designated Hosts

Qi ZHANG
University of Wollongong

Yi MU
University of Wollongong

Minji ZHANG
University of Wollongong

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

DOI: https://doi.org/10.1109/NSS.2009.59

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

Part of the Information Security Commons

Citation

ZHANG, Qi; MU, Yi; ZHANG, Minji; and DENG, Robert H.. Secure Mobile Agents with Designated Hosts. (2009). *NSS '09: Third International Conference on Network and System Security: Gold Coast, Queensland, Australia, 19-21 October 2009*. 286-293. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/633

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

2009 Third International Conference on Network and System Security
http://dx.doi.org/10.1109/NSS.2009.59

Secure Mobile Agents with Designated Hosts

Qi Zhang*, Yi Mu*, Minjie Zhang* and Robert H. Deng†
*School of Computer Science and Software Engineering
University of Wollongong
Wollongong, NSW 2522, Australia
{qz126, ymu, minjie}@uow.edu.au
†School of Information Systems
Singapore Management University
Singapore 188065
robertdeng@smu.edu.sg

***Abstract*—Mobile agents often travel in a hostile environment where their security and privacy could be compromised by any party including remote hosts in which agents visit and get services. It was proposed in the literature that the host visited by an agent should jointly sign a service agreement with the agent's home, where a proxy-signing model was deployed and every host in the agent system can sign. We observe that this actually poses a serious problem in that a host that should be excluded from an underlying agent network could also send a signed service agreement. In order to solve this problem, we propose a secure mobile agent scheme achieving host authentication with designated hosts, where only selected hosts can be included in the agent network. We also present a security model and provide a rigorous security proof to our scheme.**

***Keywords*-Authentication; Mobile agent security;**

I. Introduction

A mobile agent can travel cross an agent network performing tasks on behalf of its owner. The mobile agent technology has drawn much attention in recent years because of its potential to bringing new ways in electronic commerce. As an example, a mobile agent could be released by its owner to get the best deal from one of online sellers for finding the best offers. It can travel around the network to search and negotiate with the suitable sellers. After the deal is done, it returns the result to its owner.

Although it is generally believed that mobile agent is a powerful tool for online transactions, security and privacy is indeed a concern. A major problem is their inability to authenticate transactions in hostile environments [1]. When a mobile agent arrives at a remote host, it will be fully controlled by the host. Therefore, it is believed that it is impossible for a mobile agent to carry out any secret computation without exposing its secret to the malicious host [2], [3].

There are several mobile agent authentication schemes in the literature, where a verity of security issues were identified. Sander and Tschudin [4] concluded the following fundamental problems of executing mobile code: code and execution integrity, code privacy, and computation with a secret in public. They gave an answer to the above problems by proposing a concept called Computing with Encrypted Functions (CEF). Kotzanikolaou *et al* [1] implemented the CEF scheme and proposed an undetachable signature scheme based on RSA. Although their scheme could conceal the agent owner's private key during an execution in an untrusted environment, it does not provide the fairness of contract [5], since the remote host is not committed to the transaction whereas the customer is. Therefore, the commitment of the host is required in the transaction to prevent impersonation attack to the host.

One of the most important security services to mobile agent systems is non-repudiation, which provides fairness of transactions to hosts and agent owners. Proxy signature is thought to be an appropriate solution to the repudiation issue in mobile agent applications. We notice that several solutions derived from proxy signatures were published [6], [5], [7], [8], [9], [10]. Unfortunately, many of them are insecure [8], [11], [5], [12]. By default, a proxy-based approach such as Lee *et al* [5] grants the universal signing privilege to all hosts in the agent network; that is, any host can generate a valid signature by executing a mobile agent. In other words, a non-repudiation service agreement between the agent owner and any host can be reached out of the control of the agent owner. This assumption would be fine, if the agent owner wants to receive services from all hosts in the agent network including the undesirable ones.

We observe that an agent owner might not regard that all the hosts in the agent network are desirable for a designated service. For instance, the agent owner wants to get services such as mortgage information from some specified banks only. Therefore, hosts that do not belong to these banks should be excluded from the network. This raises a challenging question: how to construct a mobile agent network that includes only the designated hosts?

In this paper, we provide a sound answer to the question by introducing a novel scheme, which allows designated hosts to perform an agent task. The list of designated hosts can be chosen by the agent owner. Our scheme can be

regarded as a policy-based scheme, in that the agent can carry a tamper-resistant policy data set when implementing a task, where the hosts defined in the policy are included in the specific network and can provide services. One of features in our scheme is that the policy can be dynamically updated for various tasks without any additional computation cost. This feature inherits the elegant property from the dynamic accumulator [13], [14], [15], in that it does not increase the size of policy data set when the policy is changed. Our scheme also inherits the merit from proxy-based approaches that the repudiation issue is eliminated. We also define a rigorous security model for mobile agent transaction, which captures the most powerful attacks including adaptive-chosen-message and adaptive-chosen-host. These types of attacks are not captured in the existing schemes in the literature. The security of our scheme is based on the hardness of Computational Diffie-Hellman problem in the random oracle model.

The rest of this paper is organized as follows. In Section 2, we review the typical mobile agent architecture and the security issues of mobile agents. In Section 3, we introduce our mobile agent architecture and the transaction procedure. We then provide our security model in Section 4. In Section 5, we present our scheme, followed by the security proof of our scheme in Section 6. We conclude this paper in Section 7.

II. Mobile Agents and Security Issues

The typical architecture of a mobile agent application in online transactions, e-trading and agent-based information retrieval systems includes three main parties: customer, mobile agent, and remote host. Customer is the owner of mobile agent. With a task in mind, the customer generates a suitable mobile agent and delegates the task to it. The mobile agent travels in the defined agent network searching for the remote host(s) that can provide a suitable service. After a suitable host is found, the agent is executed in the host and interacts with the host. If the execution is successful, the mobile agent then returns to the customer side with the results of the execution. An agent might travel to several hosts before finding a desirable result.

The mobile agent paradigm extends the capabilities of traditional ways of remote communication and distributed computing, but unfortunately raises new security issues. The protection of an agent system is generally referred to as two aspects: protection of an agent host and protection of a mobile agent.

It is generally believed that protecting an agent host from attacks by a malicious customer or a mobile agent can be easily achieved [4], [1]. A more challenging problem is how to protect a mobile agent from being abused by a malicious host. Since the mobile agent is executed in the host, the host has access to the mobile agent code. Any unprotected information embedded in a mobile agent can be potentially leaked to the host. In this regard, the integrity of mobile agents is a major concern. Integrity protection of mobile agents against malicious host can be divided into two main categories [1]: detection and prevention.

A detection approach traces the identity of the illegitimate transaction and misbehavior. The tracing mechanism will reveal the malicious host after the illegal behavior happened. However, there are many cases indicating that this kind of solution is not sufficient. A prevention approach provides an active manner. The mobile agent is able to detect whether the host is malicious before being executed. It can only be executed after the host passes authentication. Otherwise, the execution request will be denied. The mobile agent also has the ability to conceal against the host.

Apart from the issues outlined above, fairness in an agent transaction is also a major concern. Transaction fairness can be referred to as the non-repudiation service to agent transactions. In other words, the host should not be able to deny an offer that has been promised and the customer should not be able to deny a service he wishes to obtain. As outlined in the introduction section, the non-repudiation service can be provided with a suitable proxy-based approach. Unfortunately, this kind of approach is undesirable for the case where some specific hosts must be excluded from an agent network. The motivation of this paper is to provide a sound solution to this issue.

III. Task execution procedure of proposed mobile agent

In general, the major procedures of our mobile agent system for executing a task in online applications consist of the following phases: Customer Setup, Agent Setup, Agent Dispatch, Host Execution, and Verification.

- **Customer Setup:** The customer decides the services he intends to receive and selects a set of hosts for inclusion.
- **Agent Setup:** The customer generates a delegation token based on its requirement and embeds it in the mobile agent. This token includes the list of designated hosts that are permitted in the agent network.
- **Agent Dispatch:** The mobile agent travels around the network and searches for host from the list.
- **Host Execution:** When a mobile agent arrives at a host, the host checks the validity of the delegation token. If it is invalid, the host will stop execution; otherwise, it will execute the mobile agent following the designated procedure.
- **Verification:** Anyone can verify whether the signed service is valid, following the verification algorithm.

Figure 1 demonstrates an example of a mobile agent executing a task on behalf of its owner in a general online application. The customer, as the initiator, selects a set of hosts (1, 2, and 5) for inclusion. The customer then generates a delegation token based on the task and the designated host

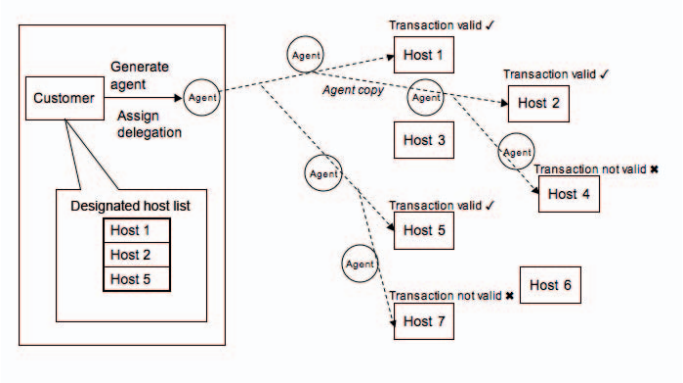

Figure 1. Agent-based Online Application Architecture

list, and embeds the token in a mobile agent. Mobile agent then travels around the network searching for the designated hosts from the list. When an agent arrives at a host, say Host 1, Host 1 verifies the delegation token prior to an execution. Because Host 1 is in the list, it can be validated and offer a signed service agreement satisfying the task defined by the customer. Hosts 3, 4, 6, and 7 are excluded from the agent network, because they are not defined in the delegation token.

IV. Security Models and Preliminaries

We consider two types of adversary: (1) malicious hosts and (2) malicious customer. Type (1) are classified into two subtypes: (a) malicious hosts that are in the designated host list and (b) malicious hosts that are not in the designated host list. To prove security in our system, we will show that, for (a), malicious hosts are not able to generate a valid signed service under the delegation token which is unknown to them; and for (b), malicious hosts possess the delegation token, but they are not able to generate a valid signed service.

A. Existential Unforgeability Against Malicious Hosts

In this case, we assume that the adversary can query the private key of any host it chooses, but cannot query the private key of the customer. We allow all the malicious hosts to collude, i.e., the adversary can obtain all private keys of the hosts in the agent network.

1) Malicious hosts that are in the designated host list: Here, we will show that, the probability of an adversary $\mathcal{A}_1$ to output a valid signature under the delegation token that is unknown to him is negligible. It is defined using the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}_1$:

- **Setup**: $\mathcal{C}$ runs the $\mathsf{ParaGen}$ algorithm to obtain system's parameters $Param$.
- **Key extract queries**: Given an identity ID of a host chosen by $\mathcal{A}_1$, $\mathcal{C}$ returns the private key s_{ID} corresponding to ID.
- **Delegation queries**: $\mathcal{A}_1$ can choose the warrant w adaptively and the designated host list $X = (H(\mathsf{ID}_1), H(\mathsf{ID}_2), ..., H(\mathsf{ID}_k))$ adaptively, and submit these to $\mathcal{C}$. Here, $k \leq \theta$ and θ is the upper bound defined in $Param$. In response, $\mathcal{C}$ runs the algorithm $D \leftarrow \mathsf{DeleGen}(w, X)$ and returns D to $\mathcal{A}_1$. D is the delegation token of combination of X and w chosen by $\mathcal{A}_1$.
- **Signature queries**: Proceeding adaptively, $\mathcal{A}_1$ can request the signature of (m, w, X, ID_i), where m is the message (which is the offer in our case) that the host needs to sign, w is the customer's warrant, X represents the designated host list, ID_i is the identity of the host who signs the message, and ID_i is in the designated host list. In response, $\mathcal{C}$ runs the algorithm to get the delegation $D \leftarrow \mathsf{DeleGen}(w, X)$, then runs the algorithm $\sigma \leftarrow \mathsf{Sign}(s_{\mathsf{ID}_i}, m, D)$, and returns σ to $\mathcal{A}_1$.
- **Output**: Finally, $\mathcal{A}_1$ outputs $(m^*, w^*, X^*, \mathsf{ID}^*, \sigma^*)$ and wins the game if:
  1) ID^* is an identity of a host which is in the designated host list X^*.
  2) (w^*, X^*) has not been requested as one of the **Delegation queries**.
  3) $(m^*, w^*, X^*, \mathsf{ID}^*)$ has not been requested as one of the **Signature queries**.
  4) $Verify(Param, PK_C, PK_{\mathsf{ID}^*}, m^*, w^*, X^*, \mathsf{ID}^*, \sigma^*) = \mathsf{valid}$.

We define $\mathsf{Succ}_{\mathcal{A}_1}$ to be the probability that the adversary $\mathcal{A}_1$ wins the above game.

Definition 1: We say an adversary $\mathcal{A}_1$ can $(t, q_{H^*}, q_{KE}, q_D, q_S, \epsilon)$-break this scheme if $\mathcal{A}_1$ runs in time at most t, $\mathcal{A}_1$ makes at most q_{H^*} hash queries, at most q_{KE} key extract queries, at most q_D delegation queries, and at most q_S signature queries, $\mathsf{Succ}_{\mathcal{A}_1}$ is at least ϵ.

2) Malicious hosts that are not in the designated host list: In this situation, the goal of an adversary $\mathcal{A}_2$ is to output a valid signature with the host that is not in the designated host list. We will show that, even $\mathcal{A}_2$ possesses the delegation token, the probability of $\mathcal{A}_2$ to output a valid signature under the delegation token is still negligible. It is defined using the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}_2$. After all the queries:

- **Output**: $\mathcal{A}_2$ outputs $(m^*, w^*, X^*, \mathsf{ID}^*, \sigma^*)$ and wins the game if:
  1) ID^* is an identity of a host which is not in the designated host list X^*.
  2) $Verify(Param, PK_C, PK_{\mathsf{ID}^*}, m^*, w^*, X^*, \mathsf{ID}^*, \sigma^*) = \mathsf{valid}$.

We define $\mathsf{Succ}_{\mathcal{A}_2}$ to be the probability that the adversary $\mathcal{A}_2$ wins the above game.

Definition 2: We say an adversary $\mathcal{A}_2$ can $(t, q_{H^*}, q_{KE}, q_D, q_S, \epsilon)$-break this scheme if $\mathcal{A}_2$ runs in time at most t, $\mathcal{A}_2$ makes at most q_{H^*} hash queries, at most q_{KE} key

extract queries, at most q_D delegation queries, and at most q_S signature queries, $\mathsf{Succ}_{\mathcal{A}_2}$ is at least ϵ.

B. Existential Unforgeability Against Malicious Customer

We assume that a malicious customer possesses the private keys sk_C and t. We want to show that a malicious customer cannot generate a valid signature for a host. Given a valid signature, the host cannot deny the fact that he has signed the message (transaction or service). It is defined using the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}_3$. After all the queries:

- **Output**: $\mathcal{A}_3$ outputs $(m^*, w^*, X^*, \mathsf{ID}^*, \sigma^*)$ and wins the game if:
  1) ID^* is an identity of a host which is in the designated host list X^*.
  2) $(m^*, w^*, X^*, \mathsf{ID}^*)$ has not been requested as one of the **Signature queries**.
  3) $Verify(Param, PK_C, PK_{\mathsf{ID}^*}, m^*, w^*, X^*, \mathsf{ID}^*, \sigma^*) = \mathsf{valid}$.

We define $\mathsf{Succ}_{\mathcal{A}_3}$ to be the probability that the adversary $\mathcal{A}_3$ wins the above game.

Definition 3: We say an adversary $\mathcal{A}_3$ can $(t, q_{H^*}, q_S, \epsilon)$-break this scheme if $\mathcal{A}_3$ runs in time at most t, $\mathcal{A}_3$ makes at most q_{H^*} hash queries, and at most q_S signature queries, $\mathsf{Succ}_{\mathcal{A}_3}$ is at least ϵ.

C. Bilinear Mapping and Complexity Definitions

Bilinear Mapping: $\mathbb{G}_1$ is a cyclic additive group of prime order p with generator P. $\mathbb{G}_T$ is a cyclic multiplicative group with the same order p. $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ is a bilinear pairing with the following properties:

1) *Bilinearity*: for all $a, b \in \mathbb{Z}_p$, $\hat{e}(aP, bP) = \hat{e}(P, P)^{ab}$.
2) *Non-Degeneracy*: $\hat{e}(P, P) \neq 1_{\mathbb{G}_T}$.
3) *Computability*: $\hat{e}$ is efficiently computable.

In the following, we provide the complexity definitions and assumptions used in our security proof.

Definition 4: (Computational Diffe-Hellman (CDH) on $\mathbb{G}_1$) Given P, aP, bP $\in \mathbb{G}_1$, for some unknown a, b $\in_R \mathbb{Z}_p$, compute abP $\in \mathbb{G}_1$.

Definition 5: (Computational Diffe-Hellman (CDH) Assumption on $\mathbb{G}_1$) Given P, aP, bP $\in \mathbb{G}_1$, for some unknown a, b $\in_R \mathbb{Z}_p$, the following function $\mathsf{Succ}^{\mathsf{CDH}}_{\mathcal{A},\mathbb{G}_1}$ is negligible for any polynomially bounded algorithm $\mathcal{A}$.

$$\mathsf{Succ}^{\mathsf{CDH}}_{\mathcal{A},\mathbb{G}_1} = \Pr[\mathcal{A}(P, aP, bP) = abP : a, b \in_R \mathbb{Z}_p].$$

Definition 6: (q-Strong Diffie-Hellman (q-SDH) on $\mathbb{G}_1$) Given a tuple $(P, sP, ..., s^qP)$, for a unknown $s \in_R \mathbb{Z}^*_p$, compute a pair $(c, \frac{1}{s+c}P)$ where $c \in \mathbb{Z}_p$.

Definition 7: (q-Strong Diffie-Hellman (q-SDH) Assumption on $\mathbb{G}_1$) Given a tuple $(P, sP, ..., s^qP)$, for a unknown $s \in_R \mathbb{Z}^*_p$, the following function $\mathsf{Succ}^{\mathsf{q\text{-}SDH}}_{\mathcal{A},\mathbb{G}_1}$ is negligible for any polynomially bounded algorithm $\mathcal{A}$.

$$\mathsf{Succ}^{\mathsf{q\text{-}SDH}}_{\mathcal{A},\mathbb{G}_1} = \Pr[\mathcal{A}(P, sP, ..., s^qP) = ((c, \frac{1}{s+c}P) \wedge c \in \mathbb{Z}_p)].$$

V. Our Scheme

We now construct our scheme using bilinear maps in the random oracle model.

1) **Setup:**
Select $(\mathbb{G}_1, \mathbb{G}_T)$ as bilinear groups where $|\mathbb{G}_1| = |\mathbb{G}_T| = p$ for some prime p. Let P be a generator of $\mathbb{G}_1$. Define the bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$. Select four distinct secure hash functions: H, H_0, H_1 and H_2 where $H : \{0,1\}^* \rightarrow \mathbb{Z}^*_p$, $H_0, H_1, H_2 : \{0,1\}^* \times \mathbb{G}_1 \rightarrow \mathbb{G}_1$. The system parameter is $Param = (\mathbb{G}_1, \mathbb{G}_T, p, P, \hat{e}, H, H_0, H_1, H_2)$.
2) **Customer setup**:
   - Select a random number $sk_C \in \mathbb{Z}^*_p$ as his private key and compute the corresponding public key $pk_C = sk_C P \in \mathbb{G}_1$.
   - Random select $t \in \mathbb{Z}^*_p$ and compute a tuple $T = (P, tP, t^2P, t^3P, ..., t^\theta P)$, where θ is the upper bound. e.g. the customer can only build a designated hosts list that includes k hosts where $k \leq \theta$.
   - Compute $E = tsk_C P$.
   - Publish $PK_C = (E, T, pk_C)$ as the customer's public key and keep sk_C, t as his private keys.
3) **Agent Setup**:
   - Build the designated host list:
     - The customer chooses k hosts, $\mathsf{ID}_1, \mathsf{ID}_2, ..., \mathsf{ID}_k$, and computes $X = (H(\mathsf{ID}_1), H(\mathsf{ID}_2), ..., H(\mathsf{ID}_k)) \subset \mathbb{Z}_p \setminus \{-t\}$ where ID_i is the identity of a host and $k \leq \theta$.
     - Compute $V = \prod_{i=1}^{k}(H(\mathsf{ID}_i) + t)P \in \mathbb{G}_1$.
   - The customer computes the delegation token $D = sk_C(H_0(w, V) + V)$ where $w \in \{0,1\}^*$ is his warrant, V is a value representing the designated host list.
   - The customer embeds (w, X, D) in the mobile agent.
4) **Host Execution**:
   - Select a random number $s_{\mathsf{ID}_i} \in \mathbb{Z}^*_p$ as his private key where ID_i is his identity. The public key of this host is $P_{\mathsf{ID}_i} = s_{\mathsf{ID}_i} P \in \mathbb{G}_1$.
   - Verify the delegation token by checking
     $$\hat{e}(D, P) \stackrel{?}{=} \hat{e}(H_0(w, V), pk_C)\hat{e}(V, pk_C),$$
     where $V = \prod_{i=1}^{k}(H(\mathsf{ID}_i)+t)P$ can be computed using T and X without t.
   - If invalid, the host stops the execution. Otherwise,
   - The host computes $H(\mathsf{ID}_i)$,

– if $H(\mathsf{ID}_i) \notin X$, the host stops the execution. Otherwise,
– The host randomly chooses $r \in \mathbb{Z}_p^*$ and generate the signature $\sigma = (\Sigma, W_{\mathsf{ID}_i}, R)$ where

$$\begin{cases} \Sigma = D + s_{\mathsf{ID}_i} H_1(m||w||\mathsf{ID}_i, V) + \\ \qquad rH_2(m||w||\mathsf{ID}_i, V), \\ W_{\mathsf{ID}_i} = \prod_{h \in X}^{h \neq H(\mathsf{ID}_i)} (h+t)P, \\ R = rP. \end{cases}$$

Here, $W_{\mathsf{ID}_i} = \prod_{h \in X}^{h \neq H(\mathsf{ID}_i)} (h+t)P$ can be computed using T and X without t. $m \in \{0,1\}^*$ is the message (the offer of the host for the transaction) needs to be signed.

5) **Verification**: Given $Param$, public keys PK_C, P_{ID_i}, a host's identity ID_i, a warrant w, a designated host list X, a message m, and a signature $\sigma = (\Sigma, W_{\mathsf{ID}_i}, R)$, verify that

$$\begin{aligned} \hat{e}(\Sigma, P) \quad \stackrel{?}{=} \quad & \hat{e}(H_0(w, V), pk_C) \cdot \\ & \hat{e}(W_{\mathsf{ID}_i}, H(\mathsf{ID}_i) pk_C + E) \cdot \\ & \hat{e}(H_1(m||w||\mathsf{ID}_i, V), P_{\mathsf{ID}_i}) \cdot \\ & \hat{e}(H_2(m||w||\mathsf{ID}_i, V), R). \end{aligned}$$

Here, $V = \prod_{h \in X}(h+t)P$ can be computed using X and PK_C without t. If the equation holds, σ will be accepted as a valid signed service; otherwise, rejected.

VI. Security Analysis

A. Existential Unforgeable Against Adversary $\mathcal{A}_1$

Theorem 1: If there exists an adversary $\mathcal{A}_1$ who can $(t, q_{H^*}, q_{KE}, q_D, q_S, \epsilon)$-break the proposed scheme then there exists another algorithm $\mathcal{B}$ who can use $\mathcal{A}_1$ to solve an instance of the CDH problem in $\mathbb{G}_1$ with probability

$$\mathsf{Succ}^{\mathsf{CDH}}_{\mathcal{B},\mathbb{G}_1} \geq (1 - \frac{2}{q_D + q_S + 2})^{q_D + q_S} (\frac{2}{q_D + q_S + 2})^2 \epsilon$$

in time $t + c_{(\mathbb{G}_1, \mathbb{G}_T)}(q_{H^*} + q_D + q_S + q_{KE} + 1)$. Here $c_{(\mathbb{G}_1, \mathbb{G}_T)}$ is a constant that depends on $(\mathbb{G}_1, \mathbb{G}_T)$.

Proof. Algorithm $\mathcal{B}$ is given a random instance (P, aP, bP) of the CDH problem in $\mathbb{G}_1$. Its goal is to compute abP by interacting with adversary $\mathcal{A}_1$. $\mathcal{B}$ will simulate the challenger and interact with $\mathcal{A}_1$ as described below. The hash functions H, H_0, H_1, H_2 are regarded as the random oracles during the proof.

Setup:

- Run the ParaGen algorithm to obtain the system's parameters $Param$.
- Set $sk_C = a$, therefore $pk_C = aP$.
- Maintain five lists: H-$list$, H_0-$list$, H_1-$list$, H_2-$list$ and Key-$list$ that store the results of queries to random oracles respectively. Initially, they are empty.
- Randomly select $t \in \mathbb{Z}_p^*$ and compute a tuple $T = (P, tP, t^2P, t^3P, ..., t^\theta P)$ where $\theta \in \mathbb{Z}_p^*$ is the upper bound of designated host list. Compute $E = tsk_C P = taP$.
- Return $Param$ and (E, T, pk_C) to $\mathcal{A}_1$.

H queries: Adversary $\mathcal{A}_1$ can make queries to the H oracle of the input ID_i at any time, $\mathcal{B}$ checks the H-$list$ first:

- If there exists an item (ID_i, h_i) in the list, $\mathcal{B}$ will return h_i to $\mathcal{A}_1$.
- Otherwise, $\mathcal{B}$ randomly chooses $h_i \in \mathbb{Z}_p^*$ such that there is no $(., h_i)$ entry in the list.

Then, $\mathcal{B}$ returns h_i to $\mathcal{A}_1$ and adds (ID_i, h_i) to the H-$list$.

H0 queries: Proceeding adaptively, adversary $\mathcal{A}_1$ can make queries to the H_0 oracle of the input (w_i, V_j) where $w_i \in \{0,1\}^*$ and $V_j \in \mathbb{G}_1$. For a query (w_i, V_j), $\mathcal{B}$ checks the H_0-$list$ as follows:

- If there exists an item $((w_i, V_j), h_{0ij}, c_{ij}, coin_{0ij})$ in the list, $\mathcal{B}$ will return h_{0ij} to $\mathcal{A}_1$.
- Otherwise, $\mathcal{B}$ tosses a coin $coin_{0ij} \in \{0,1\}$ such that $\Pr[coin_{0ij} = 1] = \delta$ (the value of δ will be determined later).
  – If $coin_{0ij} = 1$, $\mathcal{B}$ chooses $c_{ij} \in_R \mathbb{Z}_p^*$ and computes $h_{0ij} = bP + c_{ij}P$.
  – Otherwise, $coin_{0ij} = 0$, $\mathcal{B}$ chooses $c_{ij} \in_R \mathbb{Z}_p^*$ and computes $h_{0ij} = c_{ij}P$.

If h_{0ij} has already been in the tuple $(., h_{0ij}, ., .)$ of the H_0-$list$, then $\mathcal{B}$ chooses another c_{ij} and recomputes h_{0ij}. $\mathcal{B}$ then returns h_{0ij} to $\mathcal{A}_1$ and adds $((w_i, V_j), h_{0ij}, c_{ij}, coin_{0ij})$ to the H_0-$list$.

H1 queries: Proceeding adaptively, adversary $\mathcal{A}_1$ can make queries to the H_1 oracle of the input $(m_i, w_j, \mathsf{ID}_k, V_l)$. $\mathcal{B}$ maintains an H_1-$list$ which consists of tuples $(\Omega_{ijkl}, h_{1ijkl})$. For a query $\Omega_{ijkl} = (m_i||w_j||\mathsf{ID}_k, V_l)$, $\mathcal{B}$ checks the H_1-$list$ as follows:

- If there exists an item $(\Omega_{ijkl}, h_{1ijkl})$ in the list, $\mathcal{B}$ will return h_{1ijkl} to $\mathcal{A}_1$.
- Otherwise, $\mathcal{B}$ randomly chooses $h_{1ijkl} \in \mathbb{G}_1$ such that there is no $(., h_{1ijkl})$ entry in the list.

Then, $\mathcal{B}$ returns h_{1ijkl} to $\mathcal{A}_1$ and adds $(\Omega_{ijkl}, h_{1ijkl})$ to the H_1-$list$.

H2 queries: Proceeding adaptively, adversary $\mathcal{A}_1$ can make queries to the H_2 oracle of the input $(m_i, w_j, \mathsf{ID}_k, V_l)$. $\mathcal{B}$ maintains an H_2-$list$ which consists of tuples $(\Omega_{ijkl}, h_{2ijkl}, e_{ijkl}, coin_{2ijkl})$. For a query $\Omega_{ijkl} = (m_i||w_j||\mathsf{ID}_k, V_l)$, $\mathcal{B}$ checks H_2-$list$ as follows:

- If there exists an item $(\Omega_{ijkl}, h_{2ijkl}, e_{ijkl}, coin_{2ijkl})$ in the list, $\mathcal{B}$ will return h_{2ijkl} to $\mathcal{A}_1$.
- Otherwise, $\mathcal{B}$ tosses a coin $coin_{2ijkl} \in \{0,1\}$ such that $\Pr[coin_{2ijkl} = 1] = \delta$.
  – If $coin_{2ijkl} = 1$, $\mathcal{B}$ chooses $e_{ijkl} \in_R \mathbb{Z}_p^*$ and computes $h_{2ijkl} = e_{ijkl}P$.

– Otherwise, $coin_{2ijkl} = 0$, $\mathcal{B}$ chooses $e_{ijkl} \in_R \mathbb{Z}_p^*$ and computes $h_{2ijkl} = e_{ijkl}P - bP$.

If h_{2ijkl} has already been in the tuple $(., h_{2ijkl}, ., .)$ of the H_2-$list$, then $\mathcal{B}$ chooses another e_{ijkl} and recomputes h_{2ijkl}. After that, $\mathcal{B}$ returns h_{2ijkl} to $\mathcal{A}_1$ and adds $(\Omega_{ijkl}, h_{2ijkl}, e_{ijkl}, coin_{2ijkl})$ to the H_2-$list$.

Key extract queries: In this process, $\mathcal{A}_1$ can ask the private key of any host. For a query whose identity is ID_i, $\mathcal{B}$ checks the Key-$list$:

- If there is an item $(\mathsf{ID}_i, s_{\mathsf{ID}_i})$ in the Key-$list$, $\mathcal{B}$ will return s_{ID_i} to $\mathcal{A}_1$.
- Otherwise, $\mathcal{B}$ chooses $s_{\mathsf{ID}_i} \in_R \mathbb{Z}_p^*$.

If s_{ID_i} has already been in the tuple $(., s_{\mathsf{ID}_i})$ of the Key-$list$, then $\mathcal{B}$ chooses another s_{ID_i}. After that, $\mathcal{B}$ returns s_{ID_i} to $\mathcal{A}_1$ and adds $(\mathsf{ID}_i, s_{\mathsf{ID}_i})$ to the Key-$list$.

Delegation queries: Proceeding adaptively, $\mathcal{A}_1$ can ask at most q_D delegation queries of tuple (w_i, X_j) chosen by itself. For such a query (w_i, X_j), $\mathcal{B}$ first computes accumulator value $V_j = \prod_{h \in X_j}(h+t)P$. we assume that there is a tuple $((w_i, V_j), h_{0ij}, c_{ij}, coin_{0ij})$ in H_0-$list$ containing (w_i, V_j). $\mathcal{B}$ can make an **H0** query (w_i, V_j) if that tuple does not exist.

- If $coin_{0ij} = 0$, then $H_0(w_i, V_j) = h_{0ij} = c_{ij}P$. $\mathcal{B}$ can compute $D = a(H_0(w_i, V_j) + V_j) = ac_{ij}P + aV_j = c_{ij}pk_C + a\prod_{h \in X_j}(h+t)P = c_{ij}pk_C + \prod_{h \in X_j}(h+t)pk_C$.
- If $coin_{0ij} = 1$, $\mathcal{B}$ terminates the simulation and reports failure.

Signature queries: In this process, $\mathcal{A}_1$ can ask at most q_S signature queries of his choice. For a query $(m_i, w_j, \mathsf{ID}_k, X_l)$ where ID_k is in the designated host list X_l. $\mathcal{B}$ first computes accumulator value $V_l = \prod_{h \in X_l}(h+t)P$. We assume there exists tuples $((w_j, V_l), h_{0jl}, c_{jl}, coin_{0jl})$, $(\Omega_{ijkl}, h_{1ijkl})$, $(\Omega_{ijkl}, h_{2ijkl}, e_{ijkl}, coin_{2ijkl})$, $(\mathsf{ID}_k, s_{\mathsf{ID}_k})$ and (ID_k, h_k) in the H_0-$list$, H_1-$list$, H_2-$list$, Key-$list$ and H-$list$ respectively. Otherwise, $\mathcal{B}$ can make those queries by itself. Here, $\Omega_{ijkl} = (m_i||w_j||\mathsf{ID}_k, V_l)$.

- If $coin_{0jl} = 0$, then $H_0(w_j, V_l) = h_{0jl} = c_{jl}P$. $\mathcal{B}$ can compute $D = a(H_0(w_j, V_l) + V_l) = ac_{jl}P + aV_l = c_{jl}pk_C + a\prod_{h \in X_l}(h+t)P = c_{jl}pk_C + \prod_{h \in X_l}(h+t)pk_C$. After that, $\mathcal{B}$ can generate a valid signature as described in **Host execution** in our scheme.
- If $coin_{0jl} = 1$ and $coin_{2ijkl} = 0$, then $H_0(w_j, V_l) = bP + c_{jl}P$ and $H_2(m_i||w_j||\mathsf{ID}_k, V_l) = e_{ijkl}P - bP$. $\mathcal{B}$ can generate a valid signature by setting $R = rP = uP + aP$ where $u \in_R \mathbb{Z}_p^*$, therefore

$$\begin{aligned}
\Sigma_{ijkl} &= a(H_0(w_j, V_l) + V_l) + s_{\mathsf{ID}_k} H_1(m_i||w_j||\mathsf{ID}_k, V_l) + rH_2(m_i||w_j||\mathsf{ID}_k, V_l) \\
&= a(bP + c_{jl}P) + aV_l + s_{\mathsf{ID}_k} h_{1ijkl} + r(e_{ijkl}P - bP) \\
&= abP + c_{jl}pk_C + aV_l + s_{\mathsf{ID}_k} h_{1ijkl} + (u+a)(e_{ijkl} - b)P \\
&= abP + c_{jl}pk_C + aV_l + s_{\mathsf{ID}_k} h_{1ijkl} + ue_{ijkl}P - ubP + ae_{ijkl}P - abP \\
&= c_{jl}pk_C + \prod_{h \in X_l}(h+t)pk_C + s_{\mathsf{ID}_k} h_{1ijkl} + ue_{ijkl}P - ubP + e_{ijkl}pk_C,
\end{aligned}$$

  $W_{\mathsf{ID}_k} = \prod_{h \in X_l}^{h \neq h_k}(h+t)P$ and $R = rP$. $\mathcal{B}$ then returns $(\Sigma_{ijkl}, W_{\mathsf{ID}_k}, R)$ which is a valid signature to $\mathcal{A}_1$.
- if $coin_{0jl} = 1$ and $coin_{2ijkl} = 1$, $\mathcal{B}$ terminates the simulation and reports failure.

If $\mathcal{B}$ does not abort during all the queries, $\mathcal{A}_1$ will output a valid message-signature tuple $(m^*, w^*, X^*, \mathsf{ID}^*, \sigma^*)$ with successful probability at least ϵ. Here, $\sigma^* = (\Sigma^*, W^*, R^*)$ is a valid signature, ID^* is the identity of the host, w^* is the warrant, X^* is the designated host list and m^* is the message, such that ID^* is in the designated host list X^*, (w^*, X^*) has not been requested as one of the delegation queries, $(m^*, w^*, X^*, \mathsf{ID}^*)$ has not been requested as one of the signature queries. We assume there exists tuples $(\mathsf{ID}^*, s_{\mathsf{ID}^*})$, (ID^*, h^*), $((w^*, V^*), h_{0*}, c^*, coin_{0*})$, $((m^*||w^*||\mathsf{ID}^*, V^*), h_{1*})$ and $((m^*||w^*||\mathsf{ID}^*, V^*), h_{2*}, e^*, coin_{2*})$ in the Key-$list$, H-$list$, H_0-$list$, H_1-$list$ and H_2-$list$ respectively. Otherwise, $\mathcal{B}$ can make those queries by itself.

- If $coin_{0*} = 0$ or $coin_{2*} = 0$, $\mathcal{B}$ terminates the simulation and reports failure.
- Otherwise, $coin_{0*} = 1$ and $coin_{2*} = 1$. For this case, $H_0(w^*, V^*) = h_{0*} = bP + c^*P$, $H_2(m^*||w^*||\mathsf{ID}^*, V^*) = h_{2*} = e^*P$, $\sigma^* = (\Sigma^*, W^*, R^*)$ is a valid signature. Therefore,

$$\begin{aligned}
\Sigma^* &= a(bP + c^*P + V^*) + s_{\mathsf{ID}^*} h_{1*} + r^*e^*P \\
&= abP + c^*pk_C + aV^* + s_{\mathsf{ID}^*} h_{1*} + e^*R^*
\end{aligned}$$

  and $abP = \Sigma^* - c^*pk_C - \prod_{h \in X^*}(h+t)pk_C - s_{\mathsf{ID}^*} h_{1*} - e^*R^*$. Therefore, $\mathcal{B}$ successfully solves the given instance of the CDH problem in $\mathbb{G}_1$.

Now we will show the successful probability for $\mathcal{B}$. $\mathcal{B}$ can output abP successfully if and only if:

- $\mathcal{B}$ does not abort during the **Delegation queries**. This probability is $(1-\delta)^{q_D}$;
- $\mathcal{B}$ does not abort during the **Signature queries**. This probability is $(1-\delta^2)^{q_S}$;

- $\mathcal{A}_1$ outputs a valid signature. This probability is greater than ϵ;
- $coin_{0*} = 1$ and $coin_{2*} = 1$, this probability is δ^2.

The probability of $\mathcal{B}$ can successfully output abP is

$$\begin{aligned}\mathsf{Succ}^{\mathsf{CDH}}_{\mathcal{B},\mathbb{G}_1} &\geq (1-\delta)^{q_D}(1-\delta^2)^{q_S}\delta^2\epsilon \\ &\geq (1-\delta)^{q_D+q_S}\delta^2\epsilon,\end{aligned}$$

where $\delta = \frac{2}{q_D+q_S+2}$. It is maximized as

$$\mathsf{Succ}^{\mathsf{CDH}}_{\mathcal{B},\mathbb{G}_1} \geq (1-\frac{2}{q_D+q_S+2})^{q_D+q_S}(\frac{2}{q_D+q_S+2})^2\epsilon.$$

Algorithm $\mathcal{B}$'s running time is the same as $\mathcal{A}$'s running time plus the time it takes to respond to q_{H^*} random oracle queries, q_D delegation queries, q_S signature queries, q_{KE} key extraction queries and compute abP from σ^*. We assume each query requires at most time $c_{(\mathbb{G}_1,\mathbb{G}_T)}$ which is a constant depends on the bilinear group pair $(\mathbb{G}_1, \mathbb{G}_T)$. Hence, the total running time is at most $t + c_{(\mathbb{G}_1,\mathbb{G}_T)}(q_{H^*} + q_D + q_S + q_{KE} + 1)$. This completes the proof.

B. Existential Unforgeable Against Adversary $\mathcal{A}_2$

Theorem 2: If there exists an adversary $\mathcal{A}_2$ who can $(t, q_{H^*}, q_{KE}, q_D, q_S, \epsilon)$-break the proposed scheme, then there exists another algorithm $\mathcal{B}$ that can use $\mathcal{A}_2$ to solve an instance of the q-SDH problem in $\mathbb{G}_1$ with the probability

$$\mathsf{Succ}^{\mathsf{q\text{-}SDH}}_{\mathcal{B},\mathbb{G}_1} = \mathsf{Succ}_{\mathcal{A}_2} \geq \epsilon$$

in time $t + c_{(\mathbb{G}_1,\mathbb{G}_T)}(q_{H^*} + q_D + q_S + q_{KE} + 1)$. Here $c_{(\mathbb{G}_1,\mathbb{G}_T)}$ is a constant that depends on $(\mathbb{G}_1, \mathbb{G}_T)$.

Proof. Algorithm $\mathcal{B}$ is given a random instance $(P, xP, x^2P, ..., x^qP)$ of the q-SDH problem in $\mathbb{G}_1$. Its goal is to compute $(c, \frac{1}{c+x}P)$ by interacting with adversary $\mathcal{A}_2$. $\mathcal{B}$ will simulate the challenger and interact with $\mathcal{A}_2$ as described below. The hash functions H, H_0, H_1, H_2 are regarded as the random oracles during the proof.

Setup:

- Run the $\mathsf{ParaGen}$ algorithm to obtain the system's parameters $Param$.
- Randomly choose $sk_C \in \mathbb{Z}_p^*$ as the customer's private key, therefore $pk_C = sk_C P$.
- Maintain five lists: $H\text{-}list, H_0\text{-}list, H_1\text{-}list, H_2\text{-}list$ and $Key\text{-}list$, as in the proof of theorem 1.
- Set $T = (P, xP, x^2P, ..., x^qP)$ where $q = \theta$ which is the upper bound of the designated host list, and compute $E = sk_C xP$.
- Return $Param$ and (E, T, pk_C) to $\mathcal{A}_2$.

H queries: As in the proof of theorem 1, $\mathcal{A}_2$ requests ID_i, $\mathcal{B}$ returns h_i and adds (ID_i, h_i) to the $H\text{-}list$ if there is no such entry in the list.

H0 queries: As in the proof of theorem 1, $\mathcal{A}_2$ requests (w_i, V_j), $\mathcal{B}$ returns $h_{0ij} \in_R \mathbb{G}_1$ to $\mathcal{A}_2$ and adds $((w_i, V_j), h_{0ij})$ to the $H_0\text{-}list$ if there is no such entry.

H1 queries: As in the proof of theorem 1, $\mathcal{A}_2$ requests $\Omega_{ijkl} = (m_i||w_j||\mathsf{ID}_k, V_l)$. $\mathcal{B}$ returns $h_{1ijkl} \in_R \mathbb{G}_1$ to $\mathcal{A}_2$ and adds $(\Omega_{ijkl}, h_{1ijkl})$ to the $H_1\text{-}list$ if there is no such entry.

H2 queries: As in the proof of theorem 1, $\mathcal{A}_2$ requests $\Omega_{ijkl} = (m_i||w_j||\mathsf{ID}_k, V_l)$. $\mathcal{B}$ returns $h_{2ijkl} \in_R \mathbb{G}_1$ to $\mathcal{A}_2$ and adds $(\Omega_{ijkl}, h_{2ijkl})$ to the $H_2\text{-}list$ if there is no such entry.

Key extract queries: As in the proof of theorem 1, $\mathcal{A}_2$ requests ID_i. $\mathcal{B}$ returns s_{ID_i} to $\mathcal{A}_2$ and adds $(\mathsf{ID}_i, s_{\mathsf{ID}_i})$ to the $Key\text{-}list$ if there is no such entry.

Delegation queries: As in the proof of theorem 1, $\mathcal{A}_2$ requests (w_i, X_j) where $|X_j| \leq q$. $\mathcal{B}$ first computes V_j with the help of T and X_j, then $\mathcal{B}$ computes

$$D = sk_C(H_0(w_i, V_j) + V_j) = sk_C h_{0ij} + sk_C V_j$$

and returns it to $\mathcal{A}_2$.

Signature queries: As in the proof of theorem 1, $\mathcal{A}_2$ requests $(m_i, w_j, \mathsf{ID}_k, X_l)$ where $|X_l| \leq q$. $\mathcal{B}$ first computes V_l with the help of T and X_l, then $\mathcal{B}$ computes

$$\Sigma_{ijkl} = sk_C h_{0jl} + sk_C V_l + s_{\mathsf{ID}_k} h_{1ijkl} + r h_{2ijkl},$$

$W_{\mathsf{ID}_k} = \prod_{h\in X_l}^{h\neq H(\mathsf{ID}_k)}(h+x)P$ and $R = rP$ where $r \in_R \mathbb{Z}_p^*$. Then $\mathcal{B}$ returns $\sigma = (\Sigma_{ijkl}, W_{\mathsf{ID}_k}, R)$ to $\mathcal{A}_2$.

After all the queries, $\mathcal{A}_2$ outputs a message-signature tuple $(m^*, w^*, X^*, \mathsf{ID}^*, \sigma^*)$ with successful probability at least ϵ where ID^* is the identity of a host which is not in the designated host list X^* and $\sigma^* = (\Sigma^*, W^*, R^*)$ is a valid signature. We assume there exists tuples $(\mathsf{ID}^*, s_{\mathsf{ID}^*})$, (ID^*, h^*), $((w^*, V^*), h_{0^*})$, $((m^*||w^*||\mathsf{ID}^*, V^*), h_{1^*})$ and $((m^*||w^*||\mathsf{ID}^*, V^*), h_{2^*})$ in the $Key\text{-}list$, $H\text{-}list$, $H_0\text{-}list$, $H_1\text{-}list$ and $H_2\text{-}list$ respectively. Here, $V^* = \prod_{h\in X^*}(h+x)P$. Therefore, it satisfies

$$\begin{aligned}\hat{e}(\Sigma^*, P) = \; & \hat{e}(H_0(w^*, V^*), pk_C)\cdot \\ & \hat{e}(W^*, H(\mathsf{ID}^*)pk_C + E)\cdot \\ & \hat{e}(H_1(m^*||w^*||\mathsf{ID}^*, V^*), P_{\mathsf{ID}^*})\cdot \\ & \hat{e}(H_2(m^*||w^*||\mathsf{ID}^*, V^*), R^*).\end{aligned}$$

From this equation, we have

$$(h^* + x)W^* = V^* = \prod_{h\in X^*}(h+x)P$$

and $h^* \notin X^*$. $(h^*, \frac{1}{h^*+x}P)$ can be computed with the help of T and X^* from the above equation, therefore the given instance of the q-SDH problem is solved. The probability of $\mathcal{B}$ to solve the given instance of the q-SDH problem is the same as for $\mathcal{A}_2$ who breaks the proposed scheme. The total running time of $\mathcal{B}$ is at most $t + c_{(\mathbb{G}_1,\mathbb{G}_T)}(q_{H^*} + q_D + q_S + q_{KE} + 1)$, which is similar to the proof of theorem 1. This completes the proof.

C. Existential Unforgeable Against Adversary $\mathcal{A}_3$

Theorem 3: If there exist an adversary $\mathcal{A}_3$ who can $(t, q_{H^*}, q_S, \epsilon)$-break the proposed scheme, then there exists another algorithm $\mathcal{B}$ that can use $\mathcal{A}_3$ to solve an instance of the CDH problem in $\mathbb{G}_1$ with probability

$$\mathsf{Succ}^{\mathsf{CDH}}_{\mathcal{B},\mathbb{G}_1} \geq (1 - \frac{2}{q_S + 2})^{q_S} (\frac{2}{q_S + 2})^2 \epsilon$$

in time $t + c_{(\mathbb{G}_1,\mathbb{G}_T)}(q_{H^*} + q_S + 1)$. Here $c_{(\mathbb{G}_1,\mathbb{G}_T)}$ is a constant that depends on $(\mathbb{G}_1, \mathbb{G}_T)$.

Proof. The proof is similar to the proof of Theorem 1. It therefore is omitted.

VII. Conclusion

We proposed a secure mobile agent that allows a mobile agent owner to select remote hosts for the designated agent network and eliminates the non-repudiation and misuse problems in the proxy-based mobile agent model. We also provided a rigorous security proof, where comparing with other schemes our scheme is proved secure against the strongest adversaries. We defined the security model which captures the most powerful attacks against adaptive-chosen-message and adaptive-chosen-host in the random oracle model.

References

[1] P. Kotzanikolaou, M. Burmester, and V. Chrissikopoulos, "Secure transactions with mobile agents in hostile environments," in *Information Security and Privacy–ACISP 2000,* LNCS 1841, E. Dawson, A. Clark, and C. Boyd, Eds. Springer-Verlag, 2000, pp. 289–297.

[2] D. M. Chess, "Security issues in mobile code systems," in *Mobile Agents and Security*, ser. LNCS 1419, G. Vigna, Ed. Springer-Verlag, 1998, pp. 1–14.

[3] J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth, "Cryptographic security for mobile code," in *Security and Privacy, 2001. Proceedings. 2001 IEEE Symposium on Security and Privacy*, 2001, pp. 2–11.

[4] T. Sander and C. F. Tschudin, "Protecting mobile agents against malicious hosts," in *Mobile Agents and Security*, ser. LNCS 1419, G. Vigna, Ed. Springer-Verlag, 1998, pp. 44–60.

[5] B. Lee, H. Kim, and K. Kim, "Secure mobile agent using strong non-designated proxy signature," in *Information Security and Privacy–ACISP 2001,* LNCS 2119, V. Varadharajan and Y. Mu, Eds. Springer-Verlag, 2001, pp. 474–486.

[6] S.-H. Seo and S.-H. Lee, "A secure mobile agent system using multi-signature scheme in electronic commerce," in *Web and Communication Technologies and Internet-Related Social Issues, – HSI 2003*, ser. LNCS 2713, C. W. C. et al., Ed. Springer-Verlag, 2003, pp. 527–536.

[7] H. Kim, J. Baek, B. Lee, and K. Kim, "Secret computation with secrets for mobile agent using on-time proxy signature," in *Proc. of the 2001 Symposium on Cryptography and Information Security (SCIS 2001)*, Oiso, Japan, January 2001.

[8] H.-U. Park and I.-Y. Lee, "A digital nominative proxy signature scheme for mobile communication," in *Proc. of ICICS 2001*, ser. LNCS 2229, S. Qing, T. Okamoto, and J. Zhou, Eds. Springer Verlag, 2001, pp. 451–455.

[9] Y. Lee, H. Kim, Y. Park, and H. Yoon, "A new proxy signature scheme providing self-delegation," in *Information Security and Cryptology - ICISC 2006*, ser. LNCS 4296, M. S. Rhee and B. Lee, Eds. Springer Verlag, 2006, pp. 328–342.

[10] B. Lee, H. Kim, and K. Kim, "Strong proxy signature and its applications," in *Proc. of the 2001 Symposium on Cryptography and Information Security (SCIS 2001)*, Oiso, Japan, January 2001, pp. 603–608.

[11] J.-Y. Lee, J. H. Cheon, and S. Kim, "An analysis of proxy signatures: Is a secure channel necessary?" in *Topics in Cryptology - CT-RSA 2003*, ser. LNCS 2612, M. Joye, Ed. Springer-Verlag, 2003, pp. 68–79.

[12] G. Wang, F. Bao, J. Zhou, and R. H. Deng, "Security analysis of some proxy signatures," in *Proc. of ICISC 2003*, ser. LNCS 2971, J. Lim and D. Lee, Eds. Springer-Verlag, 2003, pp. 305–319.

[13] L. Nguyen, "Accumulators from bilinear pairings and applications," in *Proc. of CT-RSA 2005*, ser. LNCS 3376, A. Menezes, Ed. Springer Verlag, 2005, pp. 275–292.

[14] M. T. Goodrich, R. Tamassia, and J. Hasic, "An efficient dynamic and distributed cryptographic accumulator." in *ISC 2002*, ser. LNCS 2433. Springer Verlag, 2002, pp. 372–388.

[15] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Advances in Cryptology, Proc. CRYPTO 2002,* LNCS 2442. Springer Verlag, 2002, pp. 61–76.