# Chapter 4.40
# Secure Agent for E-Commerce Applications

**Sheng-Uei Guan**
*National University of Singapore, Singapore*

## INTRODUCTION

One hindrance to the widespread adoption of mobile agent technology (Johansen et al., 2002) is the lack of security. SAFER, or Secure Agent Fabrication, Evolution and Roaming, is a mobile agent framework that is specially designed for the purpose of electronic commerce (Guan & Yang, 2002, 2004; Yang & Guan, 2000; Zhu, Guan, Yang, & Ko, 2000). By building strong and efficient security mechanisms, SAFER aims to provide a trustworthy framework for mobile agents. Although such an agent transport protocol provides for the secure roaming of agents, there are other areas related to security to be addressed.

Agent integrity is one such area crucial to the success of agent technology. The integrity protection for agent code is relatively straightforward. A more complex code integrity scheme to handle code-on-demand is also proposed in Wang, Guan, and Chan (2002). Agent data, however, is dynamic in nature and will change as the agent roams from host to host. Despite the various attempts in the literature (Chionh, Guan, & Yang, 2001), there is no satisfactory solution to the problem so far. Some of the common weaknesses of the current schemes are vulnerabilities to revisit attack and illegal modification (deletion/insertion) of agent data.

## DESCRIPTION OF SADIS

SADIS has been designed based on the following assumptions:

1. Entities including agents, agent butlers, and hosts should have globally unique identification number (IDs).
2. Each agent butler and host should have a digital certificate that is issued by a trusted CA. These entities will be able to use the private key of its certificate to perform digital signatures and encryption.
3. Whereas the host may be malicious, the execution environment of mobile agents should be secure and the execution integrity of the agent can be maintained.
4. Entities involved are respecting and cooperating with the SADIS protocol.

## Key Seed Negotiation Protocol

The proposed key seed negotiation protocol defines the process for key seed negotiation and session key and data encryption key derivation. When an agent first leaves the butler, the butler generates a random initial key seed, encrypts it with the destination host's public key, and deposits it into the agent before sending the agent to the destination host. It should be noted that agent transmission is protected by the agent transport protocol (Guan and Yang, 2002), thereby protecting the system from being compromised by malicious hosts.

The key seed negotiation process is based on the Diffie-Hellman (DH) key exchange protocol (Schneier, 1996), with a variation. The agent will first generate a private DH parameter *a* and its corresponding public parameter *x*. The value *x*, together with the ID of the destination host, will be encrypted using a communication session key and sent to the agent butler.

The agent butler will decrypt the message using the same communication session key (discussed later). It, too, will generate its own DH private parameter *b* and its corresponding public parameter *y*. With the private parameter *b* and the public parameter *x* from the agent, the butler can derive the new key seed and use it for communications with the agent in the new host. Instead of sending the public parameter *y* to the agent as in normal DH key exchange, the agent butler will encrypt the value *y*, host ID, agent ID and current timestamp with the destination host's public key to get message *M*. Message *M* will be sent to the agent after encrypting with the communication session key.

$$M = E(y + \text{host ID} + \text{agent ID} + \text{timestamp}, H_{pubKey})$$

At the same time, the agent butler updates the agent's itinerary and stores the information locally. This effectively protects the agent's actual itinerary against any hacking attempts related to itinerary, thereby protecting against the data deletion attack.

When the agent receives the double-encrypted DH public parameter *y*, it can decrypt with the communication session key. Since the decrypted result *M* is parameter *y* and some other information encrypted with the destination host's public key, the current host will not be able to find out the value of *y* and thus find out the new key seed to be used when the agent reaches the destination host. It should be noted that this does not prevent the host from replacing *M* with its own version *M'* with the same host ID, agent ID, timestamp but different *y*. The inclusion of host ID, agent ID inside *M* can render such attack useless against SADIS. A detailed discussion on this attack can be found in the security analysis section.

Subsequently, the agent will store *M* into its data segment and requests the current host to send itself to the destination host, using the agent transport protocol (Guan & Yang, 2002).

On arriving at the destination host, the agent will be activated. Before it resumes normal operation, the agent will request the new host to decrypt message *M*. If the host is the right destination host, it will be able to use the private key to decrypt message *M*, and thus obtain the DH public parameter *y*. As a result, the decryption of message *M* not only completes the key seed negotiation process but also serves as a means to authenticate the destination host. Once the message *M* is decrypted, the host will verify that the agent ID in the decrypted message matches the incoming agent, and the host ID in the decrypted message matches that of the current host. In this way, the host can ensure that it is decrypting for a legitimate agent instead of some bogus agent. If the IDs in the decrypted messages match, the decrypted value of *y* is returned to the agent.

With the plain value of *y*, the agent can derive the key seed by using its previously generated private parameter *a*. With the new key seed derived, the key seed negotiation process is completed.