Secure Agent Data Integrity Shield

Sheng-Uei Guan [1] and Yang Yang
Department of Electrical and Computer Engineering
National University of Singapore
10 Kent Ridge Crescent, Singapore 119260

Abstract

In the rapidly expanding field of E-Commerce, mobile agent is the emerging technology that addresses the requirement of intelligent filtering/processing of information. This paper will address the area of mobile agent data integrity protection. We propose the use of Secure Agent Data Integrity Shield (SADIS) as a scheme that protects the integrity of data collected during agent roaming. With the use of a key seed negotiation protocol and integrity protection protocol, SADIS protects the secrecy as well as the integrity of agent data. Any illegal data modification, deletion, or insertion can be detected either by the subsequent host or the agent butler. Most important of all, the identity of each malicious host can be established. To evaluate the feasibility of our design, a prototype has been developed using Java. The result of benchmarking shows improvement both in terms of data and time efficiency.

Keywords: mobile agent, agent security, agent integrity, data integrity, electronic commerce

1. Introduction

With the extensive penetration of Internet technology in our everyday life, many new opportunities arise, especially in the field of commerce. E-Commerce, or electronic

[1] Contact Author: sg_1_1@yahoo.com

commerce, is born along with the Internet. The setting up of a virtual shop leads to an immediate presence in the electronic world for a merchant. Billions around the world will be able to view the products/services online, and purchase online. With this 'click-and-mortar' concept, there is no need for the rental of expensive shops in a prime location, nor the need for hiring sales promoters. All that is needed is a web presence on the Internet.

As technology evolves, there is more to setting up a virtual shop in the Internet. With millions of virtual shops springing up in various parts of the world, it is impossible for a customer to manually browse through all the possible shops before making a purchase decision. To address this concern, mobile agent technology is coming into the limelight [14]. A mobile agent will be able to automate certain tasks that were processed manually and make certain decisions intelligently with/without the interference of its owner. With this approach, information gathering can be performed automatically within the split of a second, and the decision making process can be more efficient and reliable.

One hindrance to the widespread adoption of mobile agent technology is the lack of security. When a mobile agent carries sensitive information and private mission to execute in a remote location, the agent owner must be assured of various issues so that the agent will not be compromised, the information carried by the agent won't be stolen, the cash credit carried by the agent wont' be misused, etc. Security will be the issue that has to be addressed carefully if mobile agent is to be used in the field of electronic commerce.

SAFER, or Secure Agent Fabrication, Evolution and Roaming, is a mobile agent framework that is specially designed for the purpose of electronic commerce [1-4]. By building strong and efficient security mechanisms, SAFER aims to provide a trustworthy framework for mobile agents, increasing trust factors to end users by providing the ability to trust, predictable performance and communication channel [17]. In [2], a secure agent transport protocol is proposed to ensure roaming security. While 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. Agent integrity refers to both agent code integrity and agent data integrity. Given the static nature of agent code, the integrity protection for agent code is relatively straightforward. More complex code integrity scheme to handle code-on-demand is also proposed in [6]. Different from agent code, agent data is dynamic in nature and will change as the agent roams from host to host. Despite the various attempts in the literature, 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. In [5], AMP was proposed to address agent data integrity, did address some of the weaknesses in the current literature. Unfortunately, the extensive use of PKI technology introduces too much overhead to the protocol. Also, AMP requires the agent to deposit its data collected to the butler before it roams to another host. While this is a viable and secure approach, our approach will provide an alternative by allowing the agent to carry the data by itself without depositing it (or the data hash) onto the butler.

Besides addressing the common vulnerabilities of current literature (revisit attack and data modification attack), SADIS also strives to achieve maximum efficiency without compromising security. It minimizes the use of PKI technology and relies on symmetric key encryption as much as possible, thus reducing the overhead introduced by the security mechanism to a minimum. In terms of data efficiency, it does not require the agent to carry any encryption key or random (for encryption key derivation) with it - some existing mechanism does require that. Instead, the data encryption key and the communication session key are both derivable from a key seed that is unique to the agent's roaming session in the current host. As a result, the butler can derive the communication session key and data encryption key directly.

Another feature in SADIS is strong security. The key seed negotiation itself is based on a variation of DH key exchange. During the negotiation, it also achieves the objective of implicit destination host authentication, and prevents the current host from getting any insight into the next key seed. Furthermore, to protect the key seed, it is never used directly as encryption key throughout the scheme. Instead, it is used to derive each session key and one-time data encryption key. Effectively, each message exchange between the agent butler and the agent is protected using a different session key. There is no meaningful relationship between each session key, making it extremely difficult for any attack on the keys.

Most of the existing researches focus on how to detect integrity compromise, but neglected the need to identify the malicious host. With SADIS, the agent butler will not only be able to detect any compromise to data integrity, but to identify the malicious host effectively.

To further illustrate SADIS, we will first discuss the related work in the literature on data integrity protection and their strengths and weaknesses. Subsequently, an overview of SADIS is described, followed by the details of the two main protocols under SADIS (Key negotiation protocol and data integrity protection protocol). After that, security analysis is performed on these two protocols. The result of implementation will be presented after the security analysis.

2. Related Work

Agent data integrity has been a topic of active research in the literature for a while. There are various techniques to protect agent integrities [16], some of them based on trusted hardware, some of them based on trusted host, and some even based on conventional contractual agreements. SADIS addresses the problem of data integrity protection via a combination of techniques such as execution tracing, encrypted payload, environmental key generation and undetachable signature.

Over the years, there have been quite a number of researches targeted for agent integrity protection in one way or another. One of the newest active researches is the security architecture by Borselius [18]. While many of the security services are still under active research, the security mechanisms for protecting agents against malicious host is describe in [19]. The paper proposes two mechanisms to protect mobile agents. The first mechanism makes use of a threshold scheme to protect mobile agents. Under the mechanism, a group of agents is dispatched to carry out the task, each agent carrying a vote. The agent votes for the best bid (under a trading scenario) independently. If more than n out of m ($m > n$) agents vote for the transaction, the agent owner will agree to the transaction. The security of this threshold mechanism is based on the probability that no more than n hosts out of m are malicious. In another word, the security is established based on probability. Different from this approach, the security of SADIS is completely based on its own merits without making any assumption about the integrity of external hosts.

While the research in [18-19] is actively underway, there are other more mature researches in the area. One of such research work on agent protection is SOMA. SOMA [11], or Secure and Open Mobile Agent, developed by University of Bologna, is a Java-based mobile agent framework that provides for scalability, openness, security in the Internet. One of the research focuses of SOMA is to protect mobile agent's data integrity. To achieve this, SOMA makes use of two mechanisms: Multi Hop (MH) Protocol and Trusted Third Party (TTP) Protocol.

The advantage of MH protocol is that it does not require any trusted third party or even the agent butler for its operation. This is a highly desirable feature for agent integrity protection protocol. Unfortunately, MH protocol does not hold well against revisit attack when the agent visits two or more collaborating malicious hosts during one roaming session [5]. This limitation indicates that the security of MH protocol is based on probability (that the agent does not visit two or more collaborating malicious hosts). If the agent visits host n and host m ($n < m$) who happen to be both malicious and collaborating, there are a number of attacks possible [5] [7].

Trusted Third Party protocol uses a different approach towards agent integrity protection. Sensitive operations (e.g., data hash calculation) are performed within a trusted environment so that the result can be certified and fully trusted. While this is definitely a secure mechanism, it does introduce significant overhead and inconvenience to the infrastructure.

As a result, TTP and MH are used in combination to provide optimal security and efficiency under SOMA. However, given the nature of MH and TTP protocols, the security of its combined use is still subjected to the probability that the agent does not visit two collaborating host between visit to TTPs. In this paper, we will propose a solution that does not base its security on probability.

Another agent system that addresses data integrity is Ajanta [8]. Ajanta is a platform for agent-based application on the Internet developed in the University of Minnesota. It makes use of an append-only container for agent data integrity protection. The main objective is to allow host to append new data to the container but prevents anyone from modifying the previous data without being detected. To achieve the objective, a checksum is calculated based on the previous checksum and the signed data from a new host. All the checksums are kept in the container for verification purpose later.

Similar to the MH protocol, such an append-only container suffers from revisit attack. If an agent visits collaborating malicious host n and m ($n < m$), host m can effectively remove the agent data from n to m without being detected. Another way to attack is to place a false set of data between host n and host m such that the data favors the malicious party. As long as the signatures for the fake data are valid, there is no way the butler can find out if the agent really visited those hosts. From these attacks on existing research, the importance of protecting agent itinerary is obvious. In SADIS, agent's itinerary is implicitly updated in the agent butler during key seed negotiation. This prevents any party from modifying the itinerary recorded on the butler and guard against all itinerary related attacks.

There is one recent research on agent data integrity protection called One-Time Key Generation System (OKGS) researched in Kwang-Ju Institute of Science and Technology, South Korea [13]. OKGS proposed an innovative approach of using a one-time data encryption key to encrypt the data provided by the host, and chain the encryption key to the hash values carried by the agent. When agents roam from host to host, each of them carry a hash value C_{i-1}. When the agent reaches host i, the host will generate two random R_1 and R_2. It will perform an XOR operation on C_{i-1} and R_1, and hash the output to product data encryption key S_i. This data encryption key will be used to encrypt the data provided by the current host i. Subsequently, it will perform another XOR operation on the data encryption key S_i and R_2. The output of the XOR operation is hashed to become the next hash value C_i. The two random R_1 and R_2 will be encrypted together with the digital signature on the data using the agent butler's public key. When the agent returns to the butler, the butler can repeat the key derivation process to derive the data encryption key.

OKGS does protect the agent data against a number of attack scenarios under revisit attack, such as data insertion attack and data modification attack to certain extent. However, it does not protect the agent against deletion attack as two collaborating malicious hosts can easily remove roaming records in-between them.

Furthermore, the use of XOR operation and two different random values are identified as a main weakness of the algorithm. Firstly, XOR operation is subjected to easy manipulation if one party has control over one of the inputs and has knowledge about the others. In this case, the host can adjust the random value in such a way that the output of the XOR operation can be exactly what it wants. As a result, the host will be able to dictate the data encryption key to be used. Similarly, the host also has full control over the next hash value. Secondly, the use of two different random values does not introduce more randomness to the algorithm. On the contrary, given the vulnerability of the XOR operation earlier, using two random values gives the host more room for manipulation over the data encryption key and hash value. For example, a host can change its encryption key after the agent has left (e.g., when the agent reaches one of host's collaborating partners). By producing a different R_1, the host can change the encryption key to a different value. And by producing a suitable R_2, the hash value chaining effect can be maintained. However, it should be pointed out that the signature algorithm does prevent the host from manipulating the encrypted data even though it can manipulate the encryption key. As a result, in case of this attack, the butler will probably find that the data provided by the host is corrupted but all chained signatures are valid. The data integrity is thus corrupted without being detected. Instead of using a random number to generate data encryption key, SADIS makes use of a negotiated key seed to generate data encryption key. The advantage of this approach is that no random value needs to be encrypted and stored with the agent. This effectively reduces one PKI operation (encrypt the random value with the butler's public key) and optimizes the agent data size (does not need to carry encrypted random value any more). In addition, with the new design in SADIS, the weaknesses related to the XOR operation and two random values are not inherited.

Inspired by OKGS's innovative one-time encryption key concept, SADIS will extend this property to the communication between agent and butler as well. Not only the data encryption key is one-time, but the communication session key as well. Using efficient hash calculations, the dynamic communication session key can be derived separately by the agent butler and the agent with minimum overhead. Despite the fact that all keys are derived from the same session-based key seed, SADIS also ensures that there is little correlation between these keys. As a result, even if some of the keys are compromised, the key seed will still remain secret.

3. Overview of SADIS

SADIS has been designed based on a number of assumptions. Firstly, entities including agents, agent butlers, and hosts, should have globally unique identification number (IDs). This ID will be used to uniquely identify each entity. Secondly, each agent butler and host should have a digital certificate that is issued by a trusted CA. These entities with digital certificate will be able to use the private key of its certificate to perform digital signatures and, if necessary, encryption. Thirdly, while the host may be malicious, the execution environment of mobile agents should be secure and the execution integrity of the agent can be maintained. The last assumption is that entities involved are respecting and cooperating with the SADIS protocol. For example, where digital signature is required, the signer should be willing to perform the signature under the protocol. Given the fact that the agent may be executing in a malicious environment, and that even if the execution integrity is maintained, the privacy of the execution may not be guaranteed, SADIS does not require the agent to carry any private key with it. In addition, SADIS does not require the agent to have a pre-determined itinerary. The agent is able to decide which host is the next destination independently.

Under SADIS, data integrity and agent-to-butler communication are protected by a session-based key seed. This key seed will be negotiated between the agent and butler every time the agent roams to a new host and will remain valid throughout the agent's visit to the host. A one-time data encryption key will be derived from the key seed to encrypt data provided by the current host. The communication between the agent and the butler will be protected by communication session key. Communication session key is also derived from the key seed using a different formula.

The proposed key seed negotiation protocol lays the foundation for the data integrity protection protocol. At the end of agent roaming, the host will provide a data set to be carried by the agent. The host will also perform a digital signature on the current data as well as the signature from the previous host using its private key. The signature can be subsequently verified whenever the agent reaches a new destination or returns to the agent butler. The details of the data signature generation and integrity verification process will be discussed in details in data integrity protection protocol section.

4. Key Seed Negotiation Protocol

The proposed key seed negotiation protocol defines the process for key seed negotiation as well as session key and data encryption key derivation.

When an agent first leaves the butler, the butler will generate a random initial key seed, encrypt it with the destination host's public key and deposit 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 [3]. Otherwise, a malicious host (man-in-the-middle) can perform an attack by replacing the encrypted key seed with a new key seed and encrypt it with the destination's public key. In this case, the agent and the destination host will not know the key seed has been manipulated. When the agent starts to communicate with the butler using the wrong key seed, the malicious host can intercept all the messages and re-encrypt them with the correct key derived from the correct key seed and forward them to the agent butler. In this way, a malicious host can compromise the whole protocol.

The key seed carried by the agent is session-based, it is valid until the agent leaves the current host. When the agent decides to leave the current host, it must determine the destination host and start the key seed negotiation process with the agent butler. The key seed negotiation process is based on the Diffie-Hellman (DH) key exchange protocol [15] 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 (derivation of communication session key will be discussed later in the section). 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.

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

At the same time, the agent butler updates the agent's itinerary and stores the information locally. Since the agent itinerary is stored locally in SADIS, it effectively protects the agent's actual itinerary against any hacking attempts related to itinerary. The protection of agent itinerary in turn, protects the agent against certain data integrity attack, namely, 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 [3].

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 (this is to prevent an attack scenario depicted in the security analysis section). 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. The agent can resume normal operation in the new host.

Whenever the agent or the butler needs to communicate with each other, the sender will first derive a communication session key using the key seed and use this communication session key to encrypt the message. The receiver can make use of the same formula to derive the communication session key from the same key seed to decrypt the message.

The communication session key K_{CSK} is derived using the formula below:

$K_{CSK} = Hash(key\_seed + host\ ID + seqNo)$

The sequence number is a running number that starts with 1 for each agent roaming session. Whenever the agent reaches a new host, the sequence number will be reset to 1. In this way, each message communicated will be encrypted using a different key. Given the varying communication session key, if one of the messages is somehow lost without being detected, the butler and agent will not be able to communicate afterwards. As a result, SADIS makes use of TCP/IP as a communication mechanism so that any loss of messages can be immediately detected by the sender. In the case of an unsuccessful message, the sender will send 'ping' messages to the recipient in plain format until the recipient or the communication channel recovers. Once the communication is re-established, the sender will resend the previous message (encrypted using the same communication session key). In this way, the agent and the butler can synchronize on communication session key calculations.

When the host provides information to the agent, the agent will encrypt the information with a data encryption key K_{DEK}. The data encryption key is derived as follows:

$K_{DEK} = Hash(key\_seed + hostID)$

The details on encryption will be discussed in the next section.

5. Data Integrity Protection Protocol

The key seed negotiation protocol lays the necessary foundation for integrity protection by establishing a session-based key seed between the agent and its butler. Agent data integrity is protected through the use of this key seed and the digital certificates of the hosts. This section will illustrate the data integrity protection protocol in details.

Our data Integrity Protection protocol is comprised of two parts: chained signature generation and data integrity verification. Chained signature generation is performed before the agent leaves the current host. The agent gathers data provided by the current host d_i and construct D_i as follows:

$D_i = E(d_i + ID_{host} + ID_{agent} + timestamp, k_{DEK})$

or,

$D_i = d_i + ID_{host} + ID_{agent} + timestamp$

The inclusion of host ID, agent ID and timestamp is to protect the data from possible replay attack, especially when the information is not encrypted with the data encryption key. For example, if the agent ID is not included in the message, a malicious host can potentially replace the data provided for one agent with that provided for a bogus agent. Similarly, if timestamp is not included into the message, earlier data provided to the same agent can be used at a later time to replace current data provided to the agent from the same host. The inclusion of the IDs of the parties involved and a timestamp essentially creates an unambiguous memorandum between the agent and the host.

Note that the construction of D_i gives the flexibility to encrypt the data or keep it in plain. As far as the agent integrity protection protocol is concerned, it does not matter whether the data is encrypted (since the data integrity is protected using chained digital signature). The individual agent butler or the agent itself can decide if the data should be encrypted. As a general rule of thumb, it is recommended that the agent encrypt data that is not required for the remaining of the roaming session for maximum security.

After constructing D_i, the agent will request the host to perform a signature on the following:

$$c_i = Sig(D_i + c_{i-1} + ID_{host} + ID_{agent} + timestamp, k_{priv})$$

where c_0 is the digital signature on the agent code by its butler.

There is some advantages with the use of chained digital signature compared to the conventional signature approach. In the scenario when a malicious host attempts to modify the data from an innocent host i and somehow manages to produce a valid digital signature c_i, the data integrity would have been broken if the digital signature is independent and not chained to each other. The independent digital signature also opens the window for host *i* modify data provided to the agent at a later time (one such scenario is the agent visits one of the host's collaborating partners later). Regardless of the message format used, so long as the messages are independent of each other, host *i* will have no problem reproducing a valid signature to the modified message. In this way, data integrity can be compromised. With chained digital signature, even if the malicious host (or host *i* itself) produces a valid digital signature after modifying the data, the new signature $c_i^{'}$ is unlikely to be the same as c_i. If the new signature is different from the original signature, as the previous signature is provided as input to the next signature, the subsequent signature verification will fail, thus detecting compromise to data integrity. The inclusion of host ID, agent ID, and timestamp prevents anyone from performing a replay attack.

When the agent reaches a new destination, the host must perform an integrity check on the incoming agent. In the design of SADIS, even if the new destination host does not perform an immediate integrity check on the incoming agent, any compromise to the data integrity can still be detected when the agent returns to the butler. The drawback, however, is that the identity of the malicious host may not be established. One design focus of SADIS is not only to detect data integrity compromise, but more importantly, to identify malicious hosts. To achieve malicious host identification, it is an obligation for all hosts to verify the incoming agent's data integrity before activating the agent for execution. In the event of data integrity verification failure, the previous host will be identified as the malicious host.

Data integrity verification includes the verification of all the previous signatures. The verification of signature c_0 ensures agent code integrity, the verification of c_i ensures data provided by host h_i is intact. If any signature failed the verification, the agent is considered compromised.

While the process to verify all data integrity may seem to incur too much overhead and somewhat redundant (e.g., why need to verify the integrity of d_1 in h_3 while host h_2 already verifies that), it is necessary to ensure the robustness of the protocol and to support the function of malicious host identification. For example, if only the signatures of the n consecutive previous hosts are verified, in the scenario when the previous n hosts happen to be all malicious and collaborating with one another, these malicious hosts can somehow produce the illusion to the next innocent host that data integrity has been maintained by creating seemingly correct signatures. As the next innocent host only verifies the previous n signatures that happen to be the creation of malicious hosts, it will get the impression that data integrity has not been compromised. Although the agent butler can eventually detect such data integrity compromise (since agent butler has to verify all signatures), but there is no way to establish the identity of malicious host(s).

6. Security Analysis

To analyze the effectiveness and reliability of SADIS, a detailed security analysis is performed subjecting SADIS to a variety of attacks. Based on the attack targets, the various attacks to SADIS can be classified into data attack, key attack, signature attack, itinerary attack, and composite attack. Composite attack refers to attacks that are combinations of two or more of the above-mentioned attacks. The security analysis will be organized according to the above classifications.

- Data Attack

Data attack refers to any attempt that aims to compromise the data carried by an agent. Compromise can be in the form of data modification, deletion, or insertion.

Let's consider the scenario of data modification where a malicious host wants to modify agent data or one of the hosts in the agent itinerary attempts to modify its own data after the agent has left. Assume the data targeted is D_i provided by host i, since the agent itinerary is protected by the butler and cannot be changed, only host i can produce a valid signature if the data were to be modified. However, even if the malicious party (or even host i itself) can produce a valid signature $c_i^{'}$ corresponding to $D_i^{'}$, since c_i is chained to the signature of the next host c_{i+1}, signature verification for host $(i+1)$ will fail. If the malicious host wants to ensure the signature verification for the next host is also successful, it has to forge the signature of the next host as well. Following similar argument, in order to perform a successful data modification attack, the malicious host must be able to forge the signatures for all hosts in the itinerary since host i. As the only way to achieve this is to obtain the private keys of all the following hosts, data modification attack is extremely difficult under SADIS.

Another way to compromise the data integrity is by inserting additional data into the agent. This includes inserting into data provided by hosts in the agent itinerary as well as inserting new hosts into the existing itinerary and fabricating data from the new host. The former scenario is the same as data modification attack. In the second scenario, the malicious host essentially needs to modify the itinerary of the agent. This will be covered in the discussion on itinerary attack later in the section.

Other than data modification and data insertion, data deletion is another form of data integrity attack. As illustrated in the discussion in related work, quite a number of the existing data integrity protocols suffer from this attack. After analyzing the root cause of the vulnerabilities, it is realized that it's extremely important to protect the agent's itinerary. Otherwise, in the case of a revisit attack, the subsequent host can easily 'restore' the agent to the state of its previous visit to one of the host's collaborator in the agent's itinerary. However, if the agent's itinerary is closely guarded by the butler, any data deletion will result in modification to the agent's itinerary and thus be detected.

- Key Attack

Besides direct attack on data integrity, a malicious host may attempt to attack the various keys in order to compromise data integrity. There are three different types of keys in SADIS. They are session-based key seed, communication session key, and data encryption key.

In SADIS, the key seed is negotiated between the agent and the butler during agent roaming process. Once the key seed is negotiated, it will be kept by the agent and the butler separately. It will not be used directly as encryption key at all. Attacks to the key seed can only target at the key seed negotiation protocol. As all communication in key seed negotiation is protected by the communication session key, we can safely rule out the possibility of any third party malicious attempts to break the protocol. We can focus on the scenario where the current host attempts to break the key exchange to obtain the key seed to be used in the subsequent host. Given the simplicity of DH key exchange, the parameters available for manipulation is the DH private parameter a in plain text and the encrypted DH public parameter from butler *y* encrypted using the destination host's public key.

Firstly, without any manipulation, the current host will not be able to complete DH key exchange to find out the new key seed. This is because the DH public parameter from butler y is encrypted using the destination host's public key. Without the private key from the destination host, no one can obtain *y* to complete the key exchange. Furthermore, as the encrypted message contains the agent ID and destination host ID, the current host won't be able to send a bogus agent carrying this encrypted *y* to the destination host for decryption.

If the current host attempts to manipulate any one or both of these parameters, it is able to manipulate the key seed derived when the agent reaches the destination host (This is because any change to *a* or *y* will change the result of key exchange, and anyone can forge the encrypted *y* since the encryption key is a public key). However, the change in key seed will be immediately detected when the agent communicates with the butler or vice versa. This attack can only change the key seed in the agent but won't be able to compromise the key seed in the butler. In order to perform a successful attack, the current host must also be able to obtain the key seed in the butler so that it can act as a middle-man subsequently to intercept and replace message communicated between the butler and the agent. Unfortunately, as illustrated earlier, there is no way the current host can find out the value of DH public parameter from butler *y*. Thus, the key seed will not be compromised.

Besides key seed, SADIS makes use of communication session key and data encryption key in the protocol. These two keys are directly derived from the session-based key seed using a hash function. In the case of communication session key, a sequence number is used in the key derivation to ensure each message communicated is encrypted with a different and unrelated communication session key. As far as any third-party host is concerned, attack to communication session key or data encryption key is equivalent to attacking the encryption key given only the cipher text. Even in the extreme case when such a key is compromised, the loss is limited to the message it encrypts. The other keys will remain in secret due to the nature of one-way hash functions.

- Signature Attack

Despite being categorized separately, signature attack is meaningless if carried out alone. Usually a malicious host would need to forge digital signature when it attempts to compromise data integrity. If data integrity is not compromised, there is no need to attack the chained signature at all. Signature related attacks due to data integrity compromise have been discussed earlier in the section.

- Itinerary Attack

At the first glance, agent itinerary may not seem highly sensitive. However, as examination of related work shows, if agent itinerary is not carefully protected, it may lead to compromise to data integrity, especially in the case of data deletion as illustrated earlier in the section. Given the importance of agent itinerary protection, SADIS employs a relatively conservative approach to protecting agent itinerary by storing the itinerary information in the butler as the agent roams. As the agent updates the butler of its next destination host as part of the key seed negotiation protocol, there is no additional overhead related to the itinerary protection mechanism. With the agent itinerary updated and stored with the agent butler, there is no way a malicious host can perform any attack on the itinerary (except, of course, if it breaks into the agent butler).

- Composite Attack

As the analysis above show, agent data integrity attack may not always target only in one area. At times, in order to perform a successful attack, more than one area is targeted simultaneously. These composite attacks have been discussed in the earlier section along with analysis on different attack targets.

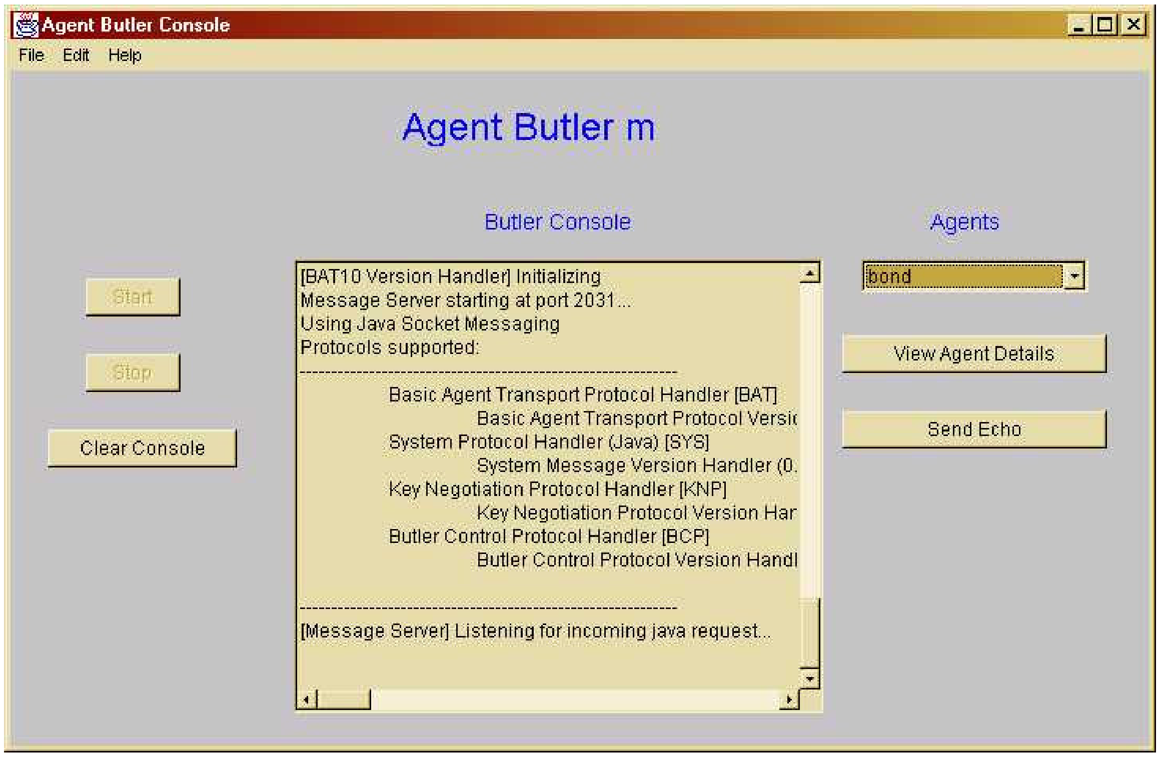
In addition to attacks with specific targets, there are certain general hacking techniques such as man-in-the-middle attack, replay attack. The design of SADIS employs a mechanism to protect the protocol against these hacking techniques. Through the use of communication session key, man-in-the-middle attack can be avoided (This is because man-in-the-middle attack will not be effective if the attacker can't decrypt the message at all). On the other hand, the use of sequence number in communication session key generation effectively protects the protocol from replay attack by a third party host. In addition, the inclusion of host ID, agent ID, and timestamp during the key seed negotiation process prevents the current host from performing a replay attack with the next destination host (attempting to obtain the next key seed).

Lastly, the design of SADIS does not have dependency on any specific encryption/hashing algorithm. In an unlikely scenario when one algorithm is broken, SADIS can always switch to a stronger algorithm.

7. Implementation

In order to verify the design of SADIS and assess its applicability, a prototype of SADIS is developed. The prototyping language is chosen to be Java. One of the main reasons for choosing Java is its platform independent feature. Internet is a complex environment that comprises of various platforms. With Java as the prototyping language, the effort required to port the prototype from one platform to another can be avoided. Furthermore, being one of the leading programming platforms in the marketplace, Java has a wide range of libraries to choose from. Modules such as cryptographic library, messaging utility, etc. are already available to be used as components in the prototype. The reuse of existing modules significantly shortened the prototyping effort, allowing the team to put its main focus on the research.

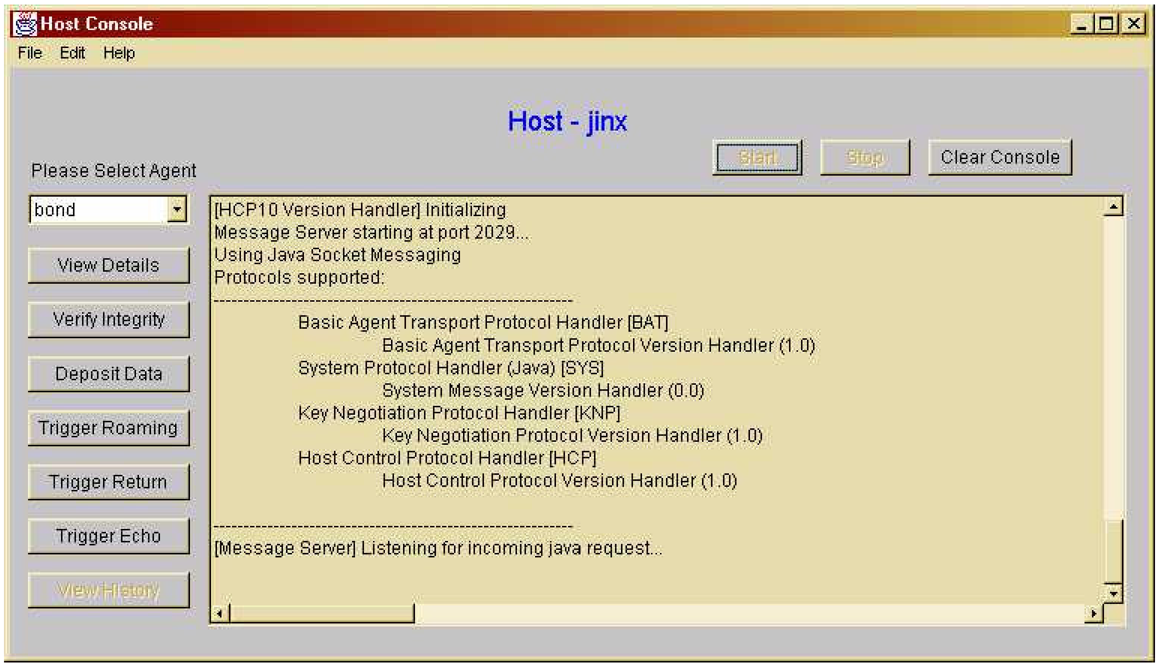
The prototype consists of four different entities: the agent butler m, agent *bond*, and two hosts *jinx* and *natalya*. The agent butler *m* (as shown in Figure 1) coordinates the agent's roaming, participates in key seed negotiation, tracks the agent's whereabouts and receives the agent during its return. Host *jinx* (as shown in Figure 2) plays the role of source host. It is the host where the agent is originally located. After agent *bond* (as shown in Figure 4) completes its processing in *jinx*, it will get *jinx* to sign the data it collected from it. Once the signature is obtained, it will trigger the key seed negotiation process with butler *m* and roam to the destination host *natalya*. Upon arrival of agent *bond*, host *natalya* (as shown in Figure 3) will perform data integrity verification on the agent *bond* before assisting it to complete the key seed negotiation process. Once the key seed negotiation is completed, agent *bond* can resume its operation. To further illustrate the use of communication session key, the agent butler *m* and the agent *bond* can send messages to each other at any time. The communication session key will be synchronized between the two, ensuring each message is encrypted using a different key. At the end of the agent roaming, agent *bond* will return to butler m. In addition to performing data integrity check on agent *bond*, *m* can also decrypt the data carried by *bond* using the various key seeds.



```
[BAT10 Version Handler] Initializing
Message Server starting at port 2031...
Using Java Socket Messaging
Protocols supported:
--------------------------------------------------------
              Basic Agent Transport Protocol Handler [BAT]
                            Basic Agent Transport Protocol Versic
              System Protocol Handler (Java) [SYS]
                            System Message Version Handler (0.
              Key Negotiation Protocol Handler [KNP]
                            Key Negotiation Protocol Version Har
              Butler Control Protocol Handler [BCP]
                            Butler Control Protocol Version Handl

--------------------------------------------------------
[Message Server] Listening for incoming java request...
```

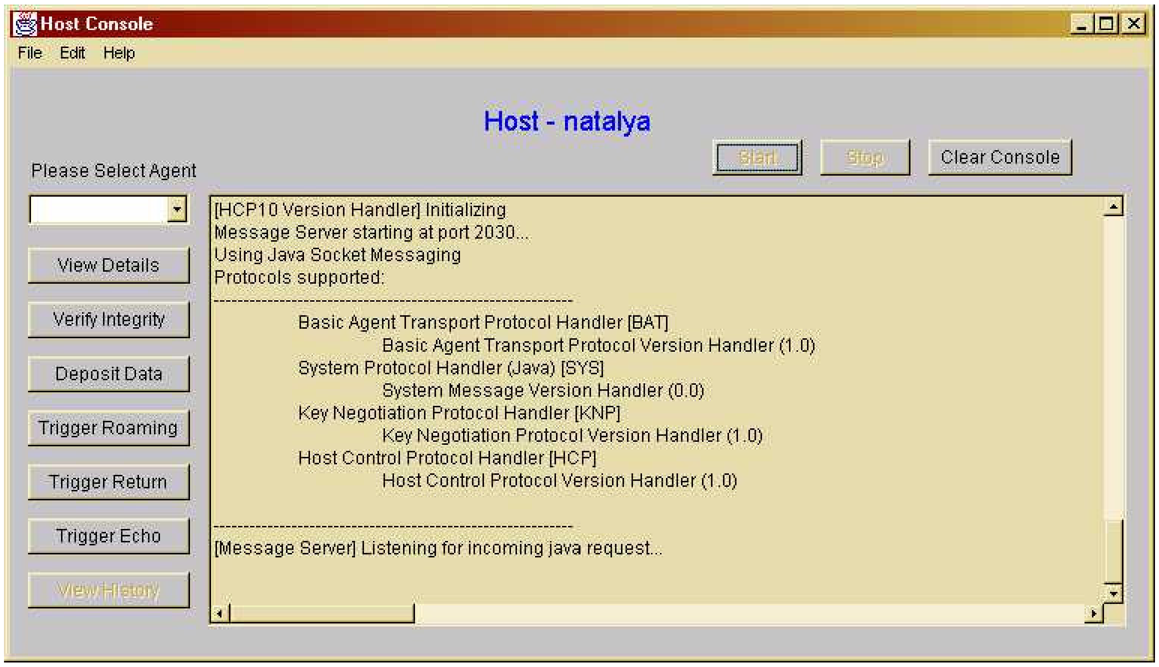
Figure 1 Agent Butler Console



```
[HCP10 Version Handler] Initializing
Message Server starting at port 2029...
Using Java Socket Messaging
Protocols supported:
------------------------------------------------------------
            Basic Agent Transport Protocol Handler [BAT]
                        Basic Agent Transport Protocol Version Handler (1.0)
            System Protocol Handler (Java) [SYS]
                        System Message Version Handler (0.0)
            Key Negotiation Protocol Handler [KNP]
                        Key Negotiation Protocol Version Handler (1.0)
            Host Control Protocol Handler [HCP]
                        Host Control Protocol Version Handler (1.0)

------------------------------------------------------------
[Message Server] Listening for incoming java request...
```

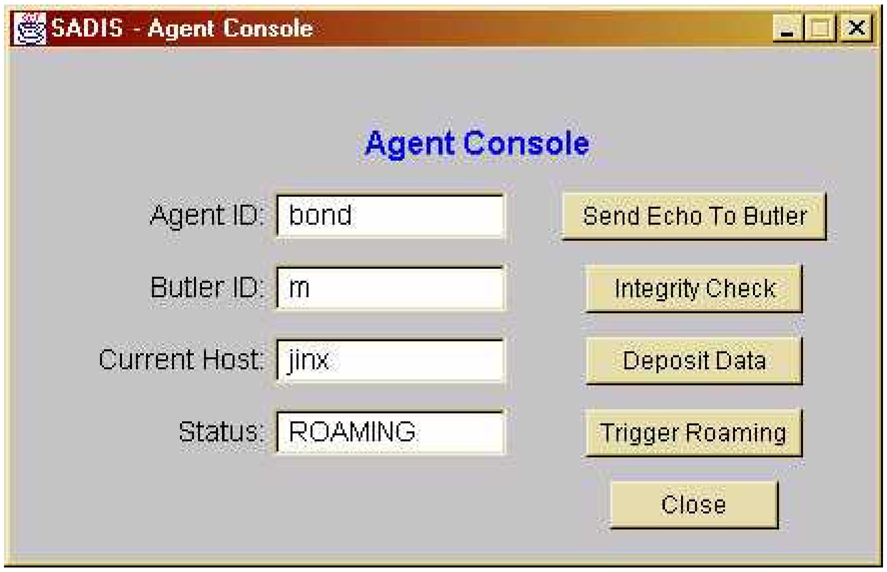
Figure 2 Host *Jinx* Console



```
[HCP10 Version Handler] Initializing
Message Server starting at port 2030...
Using Java Socket Messaging
Protocols supported:
------------------------------------------------------------
            Basic Agent Transport Protocol Handler [BAT]
                        Basic Agent Transport Protocol Version Handler (1.0)
            System Protocol Handler (Java) [SYS]
                        System Message Version Handler (0.0)
            Key Negotiation Protocol Handler [KNP]
                        Key Negotiation Protocol Version Handler (1.0)
            Host Control Protocol Handler [HCP]
                        Host Control Protocol Version Handler (1.0)

------------------------------------------------------------
[Message Server] Listening for incoming java request...
```

Figure 3 Host *Natalya* Console



Agent ID: bond
Butler ID: m
Current Host: jinx
Status: ROAMING

Figure 4 Agent *Bond*'s Console

Just like any other security mechanism, there is certain overhead associated with SADIS. The overhead is incurred as additional time required for processing as well as additional data carried by the agent.

To assess the efficiency of SADIS, a benchmarking is performed on the prototype. The benchmarking environment is composed of three PCs connected with each other via a 100MB intranet. One PC acts as the agent butler *m*, while the other two act as host *jinx* and *natalya* respectively. Agent *bond* travels between these three entities during the roaming and data collection simulation. Each PC is configured with a PIII 800 MHz processor with 512MB RAM each. The result of benchmarking SADIS is broken down based on functionality and is shown in Table 1 and Table 2. From the tables, it can be seen that the bulk of the overhead is incurred during key seed negotiation where the key exchange protocol and the public key operation is performed. During key seed negotiation, one PKI operation is incurred in the agent butler when it encrypts the public parameter of the key exchange with the destination host's public key, and another PKI operation when the destination host decrypts the incoming encrypted key exchange parameter. Given the computation intensive nature of PKI operation, it is expected that the overhead incurred during the key seed negotiation process will be relatively higher than the rest. Despite the relatively high overhead, this will not impact the overall performance of SADIS significantly because the frequency of agent roaming is low compared to the frequency of some other agent operations (such as agent to butler communication). As a result, the overhead incurred at this stage is 'one-time' in nature. Comparing with the statistics from OKGS, OKGS general incurs additional processing time of more than 500 milli-seconds. Coincidentally, this number is slightly more than twice the overhead in SADIS. Assuming there is negligible overhead caused by non-PKI operations, each PKI operation in OKGS incurs an overhead of 250 milli-seconds. In the SADIS prototype, the overhead of one PKI operation is roughly 230 milli-seconds (taking the average overhead of the key seed negotiation process). The two figures are very close to each other, suggesting a similar prototyping configuration. Considering the fact that OKGS requires one more PKI operation in the message exchange, the statistics shows the efficiency improvement of SADIS where the use of PKI operations is minimized. The time savings achieved is the time taking for one PKI operation. In the SADIS prototype, this is about 230 to 250 milli-seconds.

Operation	1 (ms)	2 (ms)	3 (ms)	4 (ms)	5 (ms)	Avg (ms)
Key Seed Negotiation (butler timing)	40	50	50	40	40	44.0
Key Seed Negotiation (destination host)	41	41	40	40	40	40.4

Agent Butler Communication (agent timing – send)	40	40	50	40	40	42.0
Agent Butler Communication (butler timing – send)	30	30	31	40	30	32.2
Agent Butler Communication (agent timing – receive)	10	10	10	10	10	10.0
Agent Butler Communication (butler timing – receive)	10	30	10	10	20	16.0

Table 1 SADIS Time Efficiency – Performance without SADIS

Operation	1 (ms)	2 (ms)	3 (ms)	4 (ms)	5 (ms)	Avg (ms)	Overhead (ms)
Key Seed Negotiation (butler timing)	250	260	250	220	260	248.0	204.0
Key Seed Negotiation (destination host)	290	281	260	280	290	280.2	239.8
Agent Butler Communication (agent timing – send)	60	60	70	50	60	60.0	18.0
Agent Butler Communication (butler timing – send)	41	50	40	40	40	42.2	10.0
Agent Butler Communication (agent timing – receive)	10	20	10	10	10	12.0	2.0
Agent Butler Communication (butler timing – receive)	30	30	30	20	20	26.0	10.0

Table 2 SADIS Time Efficiency – Performance Comparison with SADIS

Other than in the key seed negotiation, the time overhead incurred elsewhere in the protocol is negligible. As shown in the two tables, with the key seed negotiated, the time overhead incurred during message exchange will not exceed 20 milli-seconds. This is due to the use of Symmetric-Key Encryption during the more frequent message exchanges. The efficiency of the evolving communication session key can also be shown statistically as its contribution to the time overhead is negligible.

Other than overhead in terms of processing time, there is certain overhead to the data size as well. Before the detailed analysis of data overhead, it is necessary to point out that SADIS is designed to produce almost fixed data overhead regardless of the data size. In another word, regardless of the size of actual data, the overhead associated with SADIS is almost fixed, and can be limited to a fixed number of bytes. As a result, SADIS tends to be more efficient when actual data size is higher. While some of the existing literature also achieves higher efficiency when data size increase (e.g., OKGS), the size of the overhead increases when the size of actual data as well. However, the data overhead in SADIS has a maximum size regardless of the data size and does not increase as the data size increases. This ability to limit the size of overhead data regardless of actual data size is an improvement in efficiency over existing work.

The various overheads of SADIS can be best illustrated in Table 3. The first data overhead is incurred during the padding for symmetric key encryption. As most popular symmetric key encryption algorithm works on fixed length data blocks, it is necessary to pad the plain data into multiples of the block size before performing the encryption. The symmetric algorithm used in the current prototype is triple-DES that operates on blocks of 8 bytes. As a result, the padding will produce a maximum of 8 bytes data overhead.

Another data overhead is in the generation of data D_i. For security purposes, the IDs of the host and agent are added to the actual data together with the current timestamp. The prototype makes use of Java type ‘Long’ to model the IDs. And the timestamp is also a ‘Long’ in Java. Since each ‘Long’ occupies 8 bytes of storage space, the total overhead will be 24 bytes.

The last and most significant overhead is the digital signature created by the host. While the actual size of the digital signature depends on the signing algorithm used, the size of the digital signature is always a fixed length. In our prototype, RSA is used as the digital signature algorithm. Thus, the overhead of digital signature is a fixed length of 64 bytes.

Altogether, SADIS has a maximum data overhead of 96 bytes. Assuming the actual data size is 1800 bytes (this is smallest actual data size used in the benchmarking of OKGS), this yields a data overhead of 5.33%. This figure will improve linearly as the size of the actual data increases. The data overhead of 5.33% is compared with the benchmark of OKGS that averages to 36.2% (actual data size in OKGS is from 1836 to 2001).

	Original Data Size	Maximum Overhead	Overhead	OKGS Overhead
1	1800	96	5.33%	33.87%
2	2001	96	4.80%	37.73%
3	5000	96	1.92%	N/A
4	10000	96	0.96%	N/A
5	100000	96	0.10%	N/A

Table 3 SADIS Data Overhead

As the statistics shows, SADIS is optimized to improve both time efficiency and data efficiency compared with related work in the literature. The feasibility and practicality of SADIS is thus demonstrated through the prototype.

8. Conclusion

In this paper, a new data integrity protection protocol - SADIS, has been proposed. Besides being secure against a variety of attacks and robust against vulnerabilities of related work in the literature, the research of SADIS includes the objective of efficiency. This is reflected in minimized use of PKI operations and reduced message exchanges between the agent and the butler. The introduction of variation to DH key exchange and evolving communication session key further strengthened the security of the design. Unlike some existing literature, the data integrity protection protocol aims not only to detect data integrity compromise, but more importantly, to identify the malicious host.

With security, efficiency, and effectiveness as its main design focuses, SADIS works with other security mechanisms e.g. agent transport protocol to provide mobile agents with a secure platform.

References

[1] Fangming Zhu, Sheng-Uei Guan, Yang Yang, C.C. Ko, “SAFER E-Commerce: Secure Agent Fabrication, Evolution and Roaming for E-Commerce”. Electronic Commerce: Opportunities and Challenges, IDEA Group Publishing, USA, 2000.

[2] Sheng-Uei Guan, Yang Yang, “SAFE: Secure-roaming Agent For E-commerce”, Computer & Industrial Engineering Conference’99, Melbourne, Australia, pp33-37, 1999.

[3] Sheng-Uei Guan, Yang Yang, “SAFE: Secure Agent roaming for E-Commerce”, Computer & Industrial Engineering Journal, 2002.

[4] Yang Yang, Sheng-Uei Guan, “Intelligent Mobile Agents for E-Commerce: Security Issues and Agent Transport”, Electronic Commerce: Opportunities and Challenges, Idea Group Publishing, USA, 2000.

[5] Hock Boon Chionh, Sheng-Uei Guan and Yang Yang, “Ensuring the Protection of Mobile Agent Integrity: The Design of an Agent Monitoring Protocol”, Proceedings, IASTED International Conference on Advances in Communications (AIC 2001), Rhodes, Greece, pp96-99, July 2001.

[6] Tianhan Wang, Sheng-Uei Guan and Tai Khoon Chan, “Integrity Protection for Code-on-Demand Mobile Agents in E-Commerce”, Journal of Systems and Software, pp211-221, Vol. 60, Iss. 3, 2002.

[7] Volker Roth, “On the robustness of some Cryptographic Protocols for Mobile Agent Protection”, Mobile Agents 2001 (MA’01), pp1-14, 2001.

[8] Anand R. Tripathi and others, “Design of the Ajanta system for mobile agent programming”, Journal of Systems and Software, Vol. 62, Iss. 2, pp123-140, 2002.

[9] G. Karjoth, N. Asokan and C. Gulcu, “Protecting the computation results of free-roaming agents”, Mobile Agents, 1998.

[10] Walter Binder, Volker Roth, “Secure mobile agent systems using Java, where are we heading”, ACM Symposium on Applied Computing (SAC), pp115-119, 2002.

[11] A. Corradi, M. Cremonini, R. Montanari, and C. Stefanelli, “Mobile Agents and Security: Protocols for Integrity”, Proceedings of the Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems (DAIS’99), 1999.

[12] P. Bellavista, A.Corradi, and C. Stefanelli, “Protection and Interoperability for Mobile Agents: a Secure and Open Programming Environment”, IEICE Transactions on Communications, 2000.

[13] Jong-Youl Park, Dong-Ik Lee, and Hyung-Hyo Lee, “One-Time Key Generation System for Agent Data Protection”, IEICE Transactions on Information and Systems, pp535-545, 2002.

[14] Johansen, D., Lauvset, K. J., Renesse. R, Schneider, F. B., Sudmann, N.P., and Jacobsen, K., “A Tacoma Retrospective”, Software – Practice and Experience, pp605-619, 2002.

[15] B. Schneier. “Applied Cryptography: Protocols, Algorithms, and Source Code in C”, 2nd Ed., John Wiley & Sons, Inc, New York, 1996.

[16] N. Borselius, “Mobile Agent Security”, Electronics & Communication Engineering Journal, Vol. 14, no 5, IEE, London, UK, pp211-218, 2002.

[17] Andrew S. Patrick, “Building Trusthworthy Software Agents”, IEEE Journal of Internet Computing, pp46 – 53, 2002.

[18] Niklas Borselius, Namhyun Hur, Marek Kaprynski and Chris J. Mitchell. “A security architecture for agent-based mobile systems”, In Proceedings of the Third International

Conference on Mobile Communications Technologies – 3G2002, London, UK, IEE Conference Publication 489, pp312–318, 2002.

[19] N. Borselius, C.J. Mitchell and A.T. Wilson. "On mobile agent based transactions in moderately hostile environments". In B. De Decker, F. Piessens, J. Smits and E. Van Herreweghen, editors, Advances in Network and Distributed Systems Security - Proceedings of the IFIP TC11 WG11.4 First Annual Working Conference on Network Security, pp173-186. Kluwer Academic Publishers, Boston, 2001.