Performance Evaluation of Data Integrity Mechanisms for Mobile Agents*

Vandana Gunupudi
gunupudi@cs.unt.edu

Stephen R. Tate
srt@cs.unt.edu

Dept. of Computer Science and Engineering
University of North Texas
Denton, TX 76203

Abstract

A primary security challenge of the mobile agent paradigm is that of protecting the data carried by a mobile agent. With the growing popularity of e-commerce applications that use software agents, the protection of mobile agent data has become imperative. To that end, we evaluate the performance of four methods that protect the data integrity of mobile agents. While some techniques have been proposed in the literature, there has previously been no experimental study comparing the various alternatives. The set of integrity mechanisms that we investigate includes existing approaches known as the Partial Result Authentication Codes (PRACs), Hash Chaining, and Set Authentication Code methods, as well as a technique of our own design, which we refer to as the Modified Set Authentication Code method. The Modified Set Authentication Code method addresses several limitations of the Set Authentication Code method. The performance experiments were run using the DADS mobile agent system, for which we designed a Data Integrity Module. The experimental results showed that our Modified Set Authentication Code technique performed comparably to the Set Authentication Code method, showing some improvement in the key generation times and agent size. In this paper, we compare the trade-offs between security features and performance for all four methods and identify the niche that each technique could fill.

Index Terms: Data Integrity, Internet Security, Mobile Agents.

1. Introduction

The mobile agent paradigm promises to open up exciting computing possibilities, especially with the popularity of wireless networks, where bandwidth is at a premium and communication links are not always reliable. One of the emerging applications of mobile agents is in the field of electronic commerce where mobile agents can be used in buying, selling and comparison-shopping [4]. Examples of e-commerce systems include Kasbah [1] and AuctionBot [9]. These systems use software agents to provide e-commerce services to their clients. For example, in Kasbah, the users program the agent, specifying their buying/selling criteria, and the agent completes the transaction by a specified date based on the given constraints. The integrity of the mobile agent data is crucial in these settings. The user must be certain that no malicious entity tampered with any competing offers and that no unauthorized offers were added to merely influence the bidding.

In this paper, we study methods for insuring data integrity in a mobile agent setting. Our contributions include a new technique for maintaining and verifying data integrity, a common framework and notation that is used to describe all known data integrity techniques in a uniform way, and a careful experimental study of the trade-offs with various techniques.

Methods that have been devised to protect the integrity of the data carried by the agent include the Partial Result Authentication Code [10], Hash Chaining [5], and the Set Authentication Code [6] methods. Yee proposed the Partial Result Authentication Codes method [10], whereby the result of the agent's computation at each host is encapsulated using a Message Authentication Code (MAC). The result at each host, combined with the corresponding MAC, is called the Partial Result Authentication Code, or PRAC. This method requires the agent to generate a secret key (used to calculate the MAC) for each host, using a one-way function, from an initial secret key given by the originator. This method ensures that none of the results collected prior to a malicious host can be modified.

In the Hash Chaining [5] method, this idea is extended to chain the partial result to the identity of the next host to be visited. This method allows the originator to determine where exactly the chaining broke if a malicious host manipulates any of the partial results. This method, while providing stronger security, is not flexible enough to allow efficient updating of bids. New bids can be added to the agent, effectively overriding earlier bids, but the old bids must be retained in order to maintain the integrity of the entire chain.

* This research is supported in part by NSF award 0208640.

To address this problem, Loureiro, Molva, and Panetrat [7] devise an original cryptographic technique called a "Set Authentication Code," which allows the updating of an integrity proof given by the originator to the agent. Each host exchanges a secret key with the originator and uses this key to calculate the MAC on its results, and this MAC is used to update the integrity proof. The originator can efficiently verify this integrity proof upon the agent's return. Unlike hash chaining, a host that submitted an offer can subsequently change it. Because this method allows secure updating of agent data, it is useful in dynamic scenarios like online auctions.

To address some limitations of the Set Authentication Code method, we propose a method of our own design, which we call the Modified Set Authentication Code method. We propose a modification to the way secret keys are managed, which involves the use of a key that can be integrated into the set of offers carried by the agent. The secret key is encrypted with the originator's public key (carried by the agent) and is then added to the set of offers.

Each of these methods has different features and security properties. To evaluate the various methods based on different performance metrics, we have developed a data integrity module for the Distributed Agent Delivery System (DADS), a mobile agent platform developed in the Network Research Laboratory (NRL) at UNT. Using object-oriented design methodology, we have developed a general integrity interface and used sub-classing to define the data integrity methods which implement the interface.

In Section 2 we describe the basic model and security properties. In Section 3 we describe each data integrity method using a uniform notational framework. In Section 4 we describe the details of the mobile agent infrastructure used to run the comparison experiments, and in Section 5 we discuss the results of the security experiments and analyze the results. Section 6 concludes with an analysis of the trade-offs between security and performance for each method.

2. Security Model

In order to compare the data integrity methods, we use the example of a simple comparison-shopping agent, cited in mobile agent literature [5]. The agent owner (henceforth known as the originator) wishes to obtain prices from various servers (online shops) to buy a certain item. The originator programs the agent to visit different servers, querying them for the price of the item. The originator goes offline after sending the agent out, while the agent visits each host on its itinerary, collecting prices from each host, and returns to the originator with the set of competing offers. In a competitive bidding scenario, the agent may visit each host more than once. The originator will buy the item from the host that offers the lowest price (or highest desirability), based on the agent's set of offers. Above all, the originator must be assured that the set of offers is valid, i.e., each offer in the set is an authentic bid from the particular host, and that no malicious host modified any competing offer.

S_0	Originator
S_i $(i \geq 1)$	Hosts on the agent's itinerary (Intermediate hosts)
o_i	Actual offer from S_i
O_i	Encapsulated offer from S_i
$h(x)$	Cryptographic hash function
r_i	Nonce generated by S_i
$MAC_{k_i}(x)$	Message Authentication Code generated using secret key k_i

Table 1. Model Notation

2.1. Model

Each of these protocols has distinct phases: Initial setup by the originator, initial visit on intermediate hosts, updating of data, and verification of data integrity. The originator programs the agent with the information required by the first host on its itinerary. Each host is visited at least once (the initial visit). If the host offers a bid, a MAC is calculated on it and the bid is added to the agent data collection set. If a host desires to update its offer based on competing offers, it can update the offer in the agent data. Finally, the originator verifies the integrity of the agent data when it returns.

Kajorth *et al.* [5] formalized a notation for the comparison-shopping agent scenario. We combine this notation with that used by Loureiro [6, 7] in order to develop a uniform notation that can be used to formally describe each step of the protocols. Table 1 describes the resulting notation, which we use in this paper.

2.2. Security Properties

The originator requires that the agent return with valid offers. A set of security properties have been identified by Karjoth *et al.* [5], which form the core of data integrity requirements. These security properties are defined based on the assumption that a malicious host has captured an agent containing set of encapsulated offers $O_1, O_2, \ldots, O_m$, so the agent has visited m hosts where $m \leq n$. Some hosts, but not S_m, may conspire with the attacker (possibly host S_{m+1}).

- **Forward Integrity**: As defined by Yee [10], if a mobile agent visits a sequence of hosts $S_1, S_2, \ldots, S_n$, and the first malicious host is after S_m, where $1 \leq m \leq n-1$, then none of the partial results generated at hosts S_i $(i \leq m)$ can be undetectably modified by the malicious host.
- **Strong Forward integrity**: If an agent visits a set of hosts $S_1, S_2, \ldots, S_n$, and the malicious host is S_m, then none of the encapsulated offers O_k where $k > m$ can be modified.
- **Insertion Resilience**: Offers can only be added to the agent data by authorized hosts.

- **Truncation Resilience**: The chain can only be truncated at i if the host S_i colludes with the malicious host(s).

The above properties are required for a comparison-shopping agent in an interactive bidding scenario. Additional security requirements, which we don't address in this paper, include data confidentiality and non-repudiation. If a Public Key Infrastructure (PKI) is in place, these requirements are easily met.

3. Agent Data Integrity Methods

As mentioned above, each of these protocols has a common sequence of phases: Initial setup by the originator, initial visit on intermediate hosts, updating of data, and verification of data integrity. The agent visits a sequence of hosts $S_1, S_2, \ldots, S_n$, and obtains an offer o_i from each host, where an offer includes not only a price but some indication of the source of the offer[1]. Notationally, we say the agent carries a set of offers ω, a set of encapsulated offers Ω, and a key k, and we use subscripts to denote these values over time. For example, ω_i represents the value of ω after the agent has visited the ith host.

3.1. Partial Result Authentication Codes

The Partial Result Authentication Code (PRAC) method involves the calculation of a MAC on the offer at each host, using a secret key. It can be easily extended to an interactive scenario in which hosts bid against each other. Secure updates are possible using PRACs by allowing the host to retain the key used to calculate the MAC on the previous offer. The key must be removed from the agent to preserve forward integrity, but it can be retained securely by the host. If a host wants to update an offer, it simply uses the previous key and replaces the offer in the data set and the set of encapsulated offers. Upon agent return, the originator recomputes the MACs after generating the secret keys for all the visited hosts. If the MACs match, the bid is accepted.

The following formulas describe how the agent is initialized, and how it is updated as it visits a host for the first or later time.

Setup on originator:
$S_0 \rightarrow S_1$:
$\omega_0 = \emptyset$
$\Omega_0 = \emptyset$
$k_1 =$random initial key
Initial Visit on a host:
Encapsulated Offer:
$O_i = (o_i, MAC_{k_i}(o_i))$
Protocol:
$S_i \rightarrow S_{i+1}$:
$\omega_i = \omega_{i-1} \cup \{o_i\}$
$\Omega_i = \Omega_{i-1} \cup \{O_i\}$
$k_{i+1} = h(k_i)$
Update: (Changing a previous offer o_j)
Encapsulated Offer:
$O_i = (o_i, MAC_{k_j}(o_i))$
Protocol:
$S_i \rightarrow S_{i+1}$:
$\omega_i = \omega_{i-1} - \{o_j\} \cup \{o_i\}$
$\Omega_i = \Omega_{i-1} - \{O_j\} \cup \{O_i\}$
$k_{i+1} = k_i$ *(Note: New key not needed)*

The Partial Result Authentication Codes method ensures only the basic property of forward integrity whereby none of the offers from hosts visited prior to a malicious host can be modified. PRACs do not ensure insertion resilience, truncation resilience, or strong forward integrity. A serious limitation of this method is that the current secret key is incorporated into the agent, thus compromising the security of all the subsequent "secrets" used by the agent.

3.2. Hash Chaining

Kajorth *et al*. [5] extended Yee's concept to ensure *strong forward integrity*, whereby none of the offers in the agent data can be modified without detection by the originator. Each offer in the data set is *chained* to the next one using the identity of the next host to be visited by the agent. Upon agent return, for each visited host, the originator decrypts the random nonce and generates the secret keys. If the chaining relation fails at a particular point, all subsequent offers are rejected.

The initial setup is similar to that of the Partial Result Authentication Codes method. As the agent follows its itinerary, each host generates a random nonce, which serves as a token of the agent instance, and creates a key from this data and the identity of the next host to be visited by the agent.

Setup on originator:
$S_0 \rightarrow S_1$:
$\omega_0 = \emptyset$
$\Omega_0 = \emptyset$
$k_1 =$random initial key
Initial Visit on a host:
Encapsulated Offer:
$O_i = (o_i, MAC_{k_i}(r_i, o_i, S_{i+1}))$
Protocol:
$S_i \rightarrow S_{i+1}$
$R_i = ENC_0(r_i)$
$\omega_i = \omega_{i-1} \cup \{(o_i, R_i)\}$
$\Omega_i = \Omega_{i-1} \cup \{O_i\}$
$k_{i+1} = h(k_i, r_i, o_i, S_{i+1})$

Each host generates a random nonce r_i and encrypts it using the originator's public key to include with the offer and the identity of the next host. The value r_i is an input to the computation of the next secret key k_{i+1}. The term r_i prevents a host S_i from being able to modify the offers of hosts that follow it, since it cannot identify the chaining value that must be inserted into the encapsulated offer of

1 Note that while we are using a comparison shopping example and the corresponding terminology, o_i could actually represent arbitrary state information from an agent after visiting S_i in any application.

the hosts that follow it. Since the offer itself is used in computing the key for the next host, offers may not be changed, and a new key must be generated each time when the agent re-visits a host.

Update: (Changing a previous offer o_j)
Encapsulated Offer:
$O_i = (o_i, MAC_{k_i}(r_i, o_i, S_{i+1}))$
Protocol:
$S_i \rightarrow S_{i+1}$
$R_i = ENC_0(r_i)$
$\omega_i = \omega_{i-1} \cup \{(o_i, R_i)\}$
$\Omega_i = \Omega_{i-1} \cup \{O_i\}$
$k_{i+1} = h(k_i, r_i, o_i, S_{i+1})$

The chaining relation ensures that none of the offers in the agent can be modified by a malicious host, so strong forward integrity is ensured. A limited degree of insertion resilience is ensured by this protocol. A malicious server cannot insert spurious offers into the chain of hosts visited prior to it. However, it is possible for a malicious server to insert data in the chain from this point onward, since the technique explicitly allows a host to do so. In a chain of encapsulated offers $O_1, \ldots, O_m$, it is possible to detect any truncation at $k < m$, assuming that m is not malicious. However, a malicious host can truncate all the offers following its own and can insert fake offers.

One of the limitations of this technique is that verification by the originator must be performed sequentially, i.e., knowledge of the sequence of hosts visited is mandatory for verification. Also, multiple visits to a host for updates leads to a prohibitive increase in the size of the agent as old data may not be removed.

3.3. Set Authentication Codes

We present a brief description of the Set Authentication Code method. For a detailed description of the cryptographic mechanisms involved in this method, refer to [6].

Let $(k_i)_{1 \le i \le n}$ be a set of secret keys, $(o_i)_{1 \le i \le n}$ be a set of offers, and $(O_i)_{1 \le i \le n}$ be a set of encapsulated offers, with $\Omega = \{O_i | 1 \le i \le n\}$, as before. We use a function $\Gamma(\Omega)$ which gives a cryptographic "integrity proof" for set Ω, and has the following properties (for a specific function satisfying these properties, see [6]; however, note that the insert and delete notation is our own simplification, but directly maps to operations on the integrity proof in the cited paper):

- $\Gamma(\Omega)$ can not be computed without knowledge of the O_i values (or, equivalently, without knowledge of the o_i and k_i values).
- There exists an efficient function insert() such that given an integrity proof value $g = \Gamma(\{x_1, \ldots, x_k\})$ for some set $X = \{x_1, \ldots, x_k\}$ (the x_i values do not have to be known), and a new value y,

$$\mathsf{insert}(g, y) = \Gamma(X \cup \{y\}) = \Gamma(\{x_1, \ldots, x_k, y\}).$$

- There exists an efficient function delete() such that given an integrity proof value $g = \Gamma(\{x_1, \ldots, x_k\})$ for some set $X = \{x_1, \ldots, x_k\}$, and a specified value x_i,

$$\begin{aligned} \mathsf{delete}(g, y) &= \Gamma(X - \{x_i\}) \\ &= \Gamma(\{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k\}). \end{aligned}$$

- Removing or modifying an offer o_i, while maintaining the integrity of the set, requires knowledge of the corresponding secret key k_i.

The function $\Gamma()$ is the "*set authentication code*" which can be used to securely collect offers from various hosts in such a way that the set can be verified by any party that knows the keys k_i and offers o_i for all $1 \le i \le n$.

Each visited host $S_{i>0}$ (i.e., each host except the originator) exchanges a secret shared key k_i with source S_0, using the Diffie-Hellman key exchange technique. The originator, S_0, then sends the agent to visit a set of hosts $S_1, S_2, \ldots, S_n$ with an initial set integrity value Γ_0 and empty data collection list ω_0. The set authentication code function, $\Gamma()$, is maintained over the set of the MACs calculated on the offers (i.e., encapsulated offers) through the use of the insert and delete functions. This set is conceptually the same as Ω in the previous methods, but may not be carried with the agent in the set authentication code method. In this case, the single integrity proof value substitutes for the set Ω, and revealing individual encapsulated offers in Ω would allow hosts to modify the integrity proof to change bids.

Setup on originator:
$S_0 \rightarrow S_1$:
$\omega_0 = \emptyset$
$\Gamma_0 = \Gamma(\emptyset)$
Initial visit on a host:
MAC on the offer:
$k_i =$key from DH key exchange
$O_i = MAC_{k_i}(o_i)$
Protocol:
$S_i \rightarrow S_{i+1}$
$\omega_i = \omega_{i-1} \cup \{o_i\}$
$\Gamma_i = \mathsf{insert}(\Gamma_{i-1}, O_i)$
Update: (Changing a previous offer o_j)
MAC on the offer:
$O_i = MAC_{k_i}(o_i)$
Protocol:
$S_i \rightarrow S_{i+1}$
$\omega_i = \omega_{i-1} - \{o_j\} \cup \{o_i\}$
$\Gamma_i = \mathsf{insert}(\mathsf{delete}(\Gamma_{i-1}, O_j), O_i)$

None of the offers in the data set can be modified by a malicious host, since the encapsulated offer and the secret key is known only to the host and to the originator. Also, since each host must exchange a secret key with the originator using the Diffie-Hellman key exchange, no unauthorized offers can be inserted into the agent data set, assuming the original key exchange is secure. If the original key exchange is compromised (e.g., by a "man-in-the-middle" attack), unauthorized offers can be inserted into the agent data. If two hosts collude, they can replace the original data

set with a new data set from the start of the truncated path, so truncation resilience is limited. The path integrity needs to be verified to ensure complete truncation resilience.

One of the limitations of this method is that a shared, secret key is required between the originator and each host that the agent visits. To perform the integrity check verification, the originator must be in possession of all the secret keys. The agent can visit a random host but that host then needs to exchange a secret key with the originator, using the Diffie-Hellman key exchange. If the host is not known at the time of agent creation, then the originator must remain online to participate in this exchange, violating a basic tenet of autonomous agent computation. Furthermore, the basic Diffie-Hellman key exchange method is vulnerable to man-in-the-middle attacks, a fact that compromises the security of this technique.

3.4. Modified Set Authentication Codes

We propose a modification of the Set Authentication Code method, whereby each secret key is generated entirely on the visited host and is encapsulated into the agent data after being encrypted with the originator's public key. For updates of offers, a host can either re-use its previous secret key or can generate a new key for each visit, but the host must remember the last key used in order to "cancel" its previous offer. The remainder of the technique is similar to the Set Authentication Code method.

The originator S_0 sends an agent to visit a set of hosts $S_1, S_2, \ldots, S_n$ with an initial set integrity value Γ and an empty data collection list ω.

Setup on originator:
$S_0 \rightarrow S_1$:
$\omega_0 = \emptyset$
$\Gamma_0 = \Gamma(\emptyset)$
Initial visit on a host:
MAC on the offer:
k_i =random secret key
$O_i = MAC_{k_i}(o_i)$
Protocol:
$S_i \rightarrow S_{i+1}$
$K_i = ENC_0(k_i)$
$\omega_i = \omega_{i-1} \cup \{(o_i, K_i)\}$
$\Gamma_i = \mathsf{insert}(\Gamma_{i-1}, O_i)$
Update: (Changing previous offer o_j)
MAC on the offer:
k_i =secret key (random or re-use k_j)
$O_i = MAC_{k_i}(o_i)$
Protocol:
$S_i \rightarrow S_{i+1}$
$K_i = ENC_0(k_i)$
$\omega_i = \omega_{i-1} - \{(o_j, K_j)\} \cup \{(o_i, K_i)\}$
$\Gamma_i = \mathsf{insert}(\mathsf{delete}(\Gamma_{i-1}, O_j), O_i)$

None of the offers in the data set can be modified by a malicious host. Since the secret key is known only to the host and is carried in encrypted form within the agent, a malicious host cannot retrieve the secret key. However, a malicious host can insert fake offers by using a different key and encrypting that with the originator's public key, so it could masquerade as another host and add fake offers to the data set to influence the bidding. Note that this is a problem common to all methods, and that digital signatures can be used to avoid this problem if a PKI is in place. As in the Set Authentication Code method, the degree of truncation resilience offered is limited: If two hosts collude, they can replace the original data set with a new data set from the start of the truncated path.

Use of the RSA cryptosystem for secure transfer of keys to the originator leads to limitations associated with public key encryption algorithms, such as the computational inefficiency of RSA (although not any higher than the Diffie-Hellman key exchange used in the Set Authentication Code method). Therefore, the time required for encryption and decryption of the secret key is high.

4. Design and Implementation

We used the Distributed Agent Delivery System (DADS) [2] mobile agent platform developed at the Network Research Laboratory at UNT. The DADS is an extensible mobile agent system that can support multi-lingual agents.

The DADS core has a basic daemon process that listens on a standard TCP port for incoming agents. The DADS daemon processes the agents, allowing communication between the various modules and transferring the agent to the appropriate module. A module is loaded by the daemon by forking a child process, which then communicates with the parent using standard interprocess communication. A module receives input from DADS and after agent processing, the output is written back to DADS. All communication between modules is controlled by DADS.

Choosing Java as our preferred agent language, we first developed a Java module for the DADS platform. The Java module interfaces with the DADS system, while handling the loading and migration of Java agents. It uses the Java class loader mechanism to load a Java agent, which thereafter executes on its own. Java's serialization mechanism is used to retrieve the variables that store the state of the agent.

All the data integrity methods that we evaluate share certain common functionality. A common data integrity interface was defined and all the methods implement this interface. The common functionality includes creating initial data on the originator, updating the data of the agent, and verifying the integrity of the agent on the originating host. The data integrity interface encapsulates this common functionality and each method implements its own version of the interface.

5. Results and Discussion

5.1. Experimental Setup

The experiments were performed on a cluster of 7 Pentium IV, 2 GHz machines in the Computer Privacy and Security (CoPS) lab at UNT. All the machines are running RedHat Linux 8.0. One machine was designated as the originator

and the other six machines were the visited hosts, with each host being visited at least once. The bidding mechanism was varied such that each host was visited multiple times, resulting in trials in which the number of host visits was 19, 30, 44, 62, 114, 141, 186, 273 and 538. For baseline measurements, null data integrity mechanisms were applied. For all implementations of data integrity mechanisms, the HMAC-SHA1 function was used to calculate the MACs on each offer, generating a 160-bit output. 1024-bit keys were used for both the RSA [8] implementation and the Diffie-Hellman key exchange mechanism [3]. We used the basic cryptographic algorithms that are part of Sun's Java Cryptography extension (JCE), and we wrote our own Java implementation of the RSA algorithm.

Traditional Diffie-Hellman key exchange mechanism (used in the Set Authentication Code [6] method) involves active interaction between the participating hosts, which is something we would like to avoid in the agent setting. In particular, it is desirable that the agent not interact with the host until it actually visits the host. To correct this problem, we implement a version of the Diffie-Hellman, which can be done with delayed interaction, and we refer to this technique as a "Partial Diffie-Hellman Key Exchange". The originator chooses the Diffie-Hellman parameters and calculates its ephemeral Diffie-Hellman public key. The public key is encapsulated into the agent data, along with the Diffie-Hellman public parameters. As the agent arrives at each host, the host calculates its Diffie-Hellman public key and private key by retrieving the Diffie-Hellman public parameters from the agent data. Each host then calculates the secret key using the originator's public key and its own private key and encapsulates its public key into the agent data. When the agent returns with the final data set, the originator retrieves the public key of each host. It calculates the secret key for each host from its own private key and the host's public key, which completes the Diffie-Hellman key exchange.

5.2. Performance Metrics

- **Key generation time**: Time required for calculating the secret key on each visited host.
  - Partial Result Authentication Codes and Hash Chaining: This includes the time required to calculate the secret key from the previous host's secret key, using a cryptographic hash function. For the Hash Chaining method, a new key is calculated each time a host is re-visited.
  - Set Authentication Code and Modified Set Authentication Code: For the Set Authentication Code method, this is the time required to complete the Diffie-Hellman exchange. For the Modified Set Authentication Code method, this is the time required to calculate a secret key and encrypt it with the originator's public key.
- **Overall Method time**: For all methods, this is the sum of the setup time on the originator, the total time spent on visited hosts, and the verification time. Therefore, this time is the total computation and communication time from start to finish for each method.

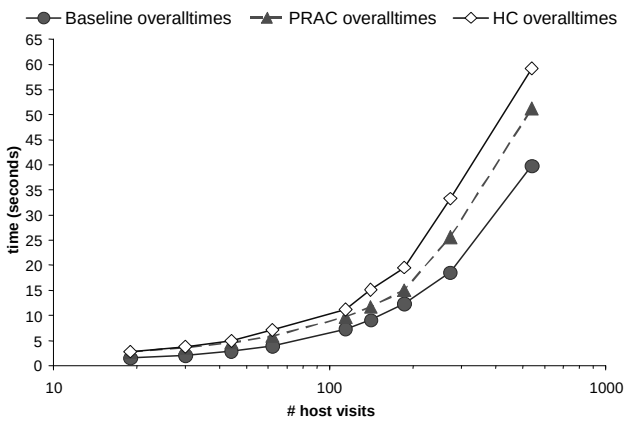

Figure 1. Overall Method Time for Partial Result Authentication Codes and Hash Chaining

5.3. Partial Result Authentication Codes v/s Hash Chaining

We compare the overall times for the Partial Result Authentication Codes and Hash Chaining methods to illustrate the trade-off between security and efficiency.

Since the chaining mechanism leads to an increase in the agent size with multiple host visits, the corresponding times to serialize/deserialize the agent are higher. Since a new key must be generated for each agent visit, the key generation times are also higher for the Hash Chaining method. The serialization times and the key generation times lead to higher overall method times for the Hash Chaining method, with the data integrity overhead (the cost over and above the baseline) for hash chaining being about twice the overhead of PRACs, as shown in Figure 1. Therefore, the additional security provided by the chaining mechanism for the Hash Chaining method is offset by the higher computation times.

5.4. Set Authentication Code v/s Modified Set Authentication Code

Since the Set Authentication Code method uses the partial Diffie-Hellman key exchange mechanism, each host must finish the key exchange during the agent's initial visit. Thus, each host retrieves the Diffie-Hellman parameters and calculates its ephemeral Diffie-Hellman public key. Then it calculates the Diffie-Hellman secret key using its own private key and the originator's ephemeral Diffie-Hellman public keys. This involves significant computation time, as seen from Figure 2.

For the Modified Set Authentication Code method, each host generates a secret key and encrypts it with the originator's RSA public key. In our experiments we reuse existing keys on subsequent visits to the same host, so only the ini-

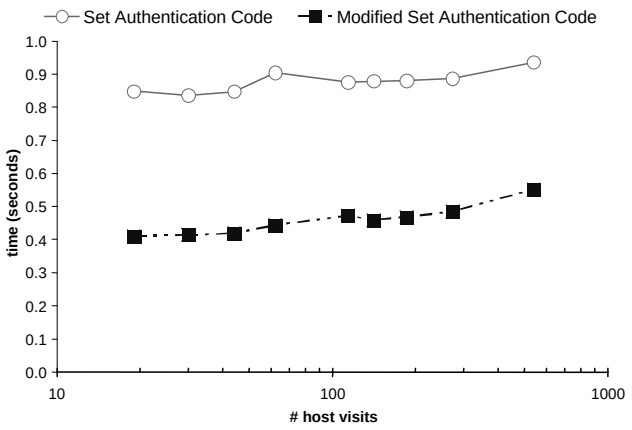

Figure 2. Comparison of Key generation time on visited hosts

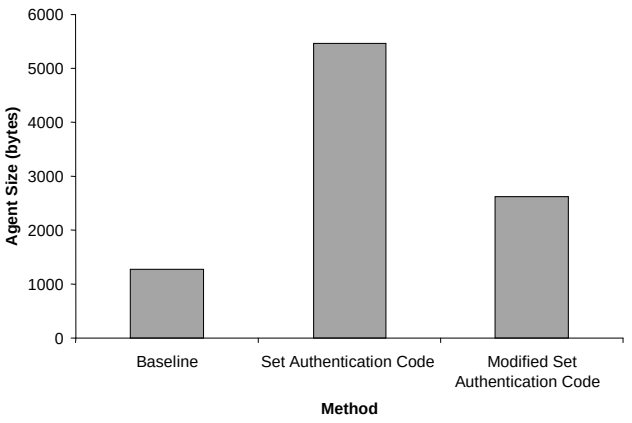

Figure 3. Final Agent size for Set Authentication Code and Modified Set Authentication Code methods

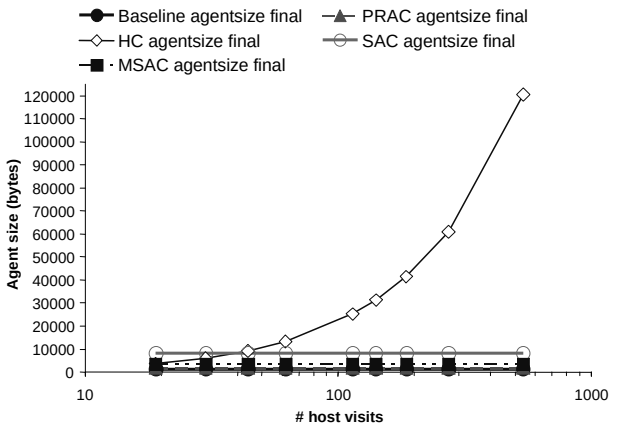

Figure 4. Agent size comparison for all methods

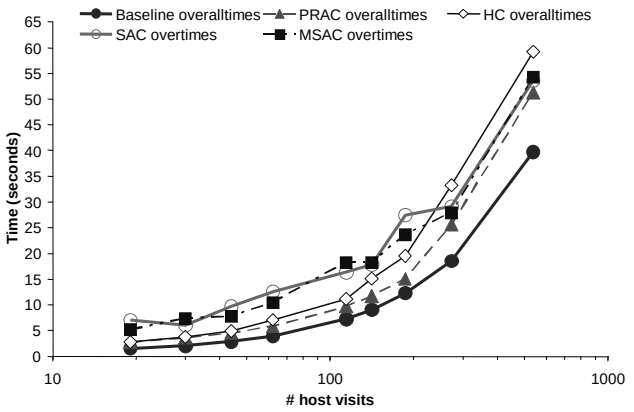

Figure 5. Overall Method times comparison for all methods

tial key generation and encryption is needed. RSA encryption is an expensive computation, but even so, our results show that the time required for the partial Diffie-Hellman key exchange is higher than the computation time for RSA encryption.

As seen in Figure 3, the final agent size is considerably smaller for the Modified Set Authentication Code method when compared to the Set Authentication Code method. The size of the agent is larger for the Set Authentication Code method since the ephemeral Diffie-Hellman public keys of all the hosts are incorporated into the agent data, as opposed to encapsulating the encrypted secret keys for the Modified Set Authentication Code method.

5.5. Comparison of All methods

Agent size is an important metric when considering the use of mobile agents in e-commerce applications. One of the limitations of using data integrity mechanisms in general, is that the size of the agent increases when cryptographic protocols are applied to protect the agent.

From Figure 4, it can be seen that the size of the agent increases dramatically with the number of host visits for the Hash Chaining method, making it less attractive for interactive-bidding scenarios. For all other methods, the agent size is a relatively minor overhead.

Figure 5 shows the overall times for all the methods. With increasing number of host visits, the overall times increase rapidly for the Hash Chaining method, overtaking even the times required for the more powerful Set Authentication Code and Modified Set Authentication Code methods. The Partial Result Authentication Codes method takes the least overall time but also offers the weakest security. The key exchange mechanism, which is a central part of the Set Authentication Code and Modified Set Authentication Code methods, leads to high overall times for these methods.

6. Security/Performance Trade-offs

We compare the various methods for two kinds of applications: a competitive bidding setting and a static bid collection setting. In the static bid collection setting, each host is only visited once, whereas in the competitive bidding setting, each host is visited multiple times, allowing updates of previous offers. An example of a static bid collection scenario is one in which the originator programs an agent to visit airline servers, collecting ticket prices from each server. A competitive bidding scenario could be an online auction that allows users to bid against each other.

In a static bid collection scenario, the encapsulation of the initial secret key as part of the agent is a serious security limitation of the Partial Result Authentication Codes method. Since agent size is not a major concern (due to limited number of hosts), the security provided by the chaining mechanism makes the Hash Chaining method very attractive. The originator can detect if a malicious host tampers with any of the offers and determine which host was responsible. The Set Authentication Code method requires that the originator remain online to perform the Diffie-Hellman key exchange with each visited host. In addition, in the absence of authentication of visited hosts, this protocol is vulnerable to man-in-the-middle attacks. Alternatively, the partial Diffie-Hellman key exchange mechanism, as implemented in this paper, can be used, allowing the originator to go offline. However, this makes the protocol more vulnerable to man-in-the-middle attacks.

The Modified Set Authentication Code method provides a secure way to exchange secret keys, without the limitations of the key exchange mechanism of the Set Authentication Code method. Strong forward integrity is still ensured; therefore, no host can retrieve competing offers. For a static bid collection scenario, insertion resilience is not a vital consideration, since no host can influence the offers of other hosts. Hence, the Modified Set Authentication Code method presents an elegant alternative to the Set Authentication Code method for application in the static bid collection scenario. However, the use of public key encryption algorithms reduces the efficiency of this method, making the Hash Chaining method (which offers comparable security features) a more viable alternative.

In the competitive bidding scenario, the security benefits of the Hash Chaining method are offset by the large agent size and the high computation times. The computation times become comparable to that of the more powerful Set Authentication Code method. The Partial Result Authentication Codes method offers weak security, providing only forward integrity. In this situation, there is a definite trade-off between performance and security. The choice of either method is dependent on which factor is more important, depending upon the application.

In the interactive bid collection setting, the Set Authentication Code method and Modified Set Authentication Code method have comparable efficiency. Improvement in agent size and key generation times on visited hosts makes the Modified Set Authentication Code method an attractive alternative. However, insertion resilience cannot be ensured with either method unless a PKI is in place to provide authentication. Therefore, for the competitive bidding setting, the Modified Set Authentication Code provides a viable alternative to the Set Authentication Code method.

7. Conclusion

From the above discussion, it can be seen that the choice of a data integrity mechanism depends primarily on the application setting. While the Hash Chaining method appears to be the most viable alternative for the static bid collection application, both the Set Authentication Code and Modified Set Authentication Code methods have features that recommend their use in the competitive bidding scenario. It should be noted, however, that the cost of the data integrity mechanisms is a noticeable overhead, when compared with the baseline. Thus, if data integrity is required, using the Modified Set Authentication Code method, which guarantees the widest flexibility and strong security, is a good generic solution.

References

[1] A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pages 75–90, 1996.

[2] C. Cozzolino. Distributed agent delivery system. Master's thesis, University of North Texas, Department of Computer Science, 2002.

[3] W. Diffi e and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[4] R. S. S. Filho. The mobile agents paradigm, 2001. ICS221 – Software Engineering Final Paper, U.C. Irvine.

[5] G. Karjoth, N. Asokan, and C. Gülcü. Protecting the computation results of free-roaming agents. In *2nd International Workshop on Mobile Agents*, volume 1477 of *Lecture Notes in Computer Science*, pages 195–207. Springer-Verlag, 1998.

[6] S. Loureiro. *Mobile Code Protection*. PhD thesis, ENST Paris / Institut Eurecom, 2001.

[7] S. Loureiro, R. Molva, and A. Pannetrat. Secure data collection with updates. *Electronic Commerce Research Journal*, 1(1/2):119–130, 2001.

[8] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[9] P. R. Wurman, M. P. Wellman, and W. E. Walsh. The Michigan Internet AuctionBot: A confi gurable auction server for human and software agents. In *Proceedings of Second International Conference on Autonomous Agents*, pages 301–308, 1998.

[10] B. Yee. A sanctuary for mobile agents. In J. Vitek and C. Jensen, editors, *Secure Internet Programming*, volume 1603 of *Lecture Notes in Computer Science*, pages 261–273. Springer-Verlag, 1999.