Proc. Second International Workshop on Mobile Agents (MA '98), K. Rothermel and F. Hohl, editors, Lecture Notes in Computer Science 1477, pages 195–207. Springer, 1998.

Protecting the computation results of free-roaming agents

G. Karjoth, N. Asokan, C. Gülcü

IBM Research Division
Zurich Research Laboratory
{gka|aso|cgu}zurich.ibm.com

Abstract. When mobile agents do comparison shopping for their owners, they are subject to attacks of malicious hosts executing the agents. We present a family of protocols that protect the computation results established by free-roaming mobile agents. Our protocols enable the owner of the agent to detect upon its return whether a visited host has maliciously altered the state of the agent, thus providing forward integrity and truncation resilience. In an environment without public-key infrastructure, the protocols are based only on a secret hash chain. With a public-key infrastructure, the protocols also guarantee non-repudiability.

1 Introduction

Mobile agents are software programs that live in computer networks, performing their computations and moving from host to host as necessary to fulfill their goals. In [8], White envisions mobile agents that make and keep appointments for their owners; provide useful information such as television schedules, traffic conditions, restaurant menus, and stock market results, and help them carry out financial transactions, from booking theater tickets, to ordering flowers, to buying and selling stock.

Electronic commerce appears to be a particularly interesting application for agents (BargainFinder, ShopBot [3], Kasbah [1], Bazaar, etc.). BargainFinder, for example, is an intelligent agent that comparison shops among Internet stores to find the best price for a compact disc, and thus has demonstrated the potential impact of agents on virtual retailing. Although all of the above agents are stationary, comparison shopping has been proposed for mobile agents as well. The principal scheme for comparison shopping, described for example in [8, 5, 7], is the following:

1. The agent owner sets up the agent with a description of the good he wishes to purchase.
2. The agent travels from the owner's site to the directory in the electronic marketplace, where it obtains the locations of all retailers that offer the good.

3. The agent visits the electronic store-front of each retailer in turn. Each store-front is a different location in the electronic marketplace. The agent gives the description of the good to the store-front and is quoted a price.
4. The agent eventually returns to its owner's location and delivers a report of its findings.

However, mobile agents are exposed to a very serious security threat: malicious hosts might endanger agents passing by. In terms of the above scenario, servers might have an incentive to subvert the computation of a visiting agent, for example by removing cheaper offers. Although denial-of-service attacks cannot be prevented, the following policy might be a strong incentive even for a malicious server to let the agent pass: *In order to submit his own offer a merchant must at least also preserve the offers of all merchants the agent has visited before it arrived at the merchant's server.* However, to enforce this policy it is necessary to be able to tell whether an agent has been changed at any server on its itinerary.

In [9], Yee proposed schemes using cryptographic hashing and digital signatures that offer practical solutions to protect the integrity of results collected by mobile agents. In particular, Yee defines the notion of *forward integrity*, which states if a mobile agent visits a sequence of servers $S_1, S_2, \ldots, S_n$, and the first malicious server is S_k, then none of the partial results generated at servers S_i, where $i < k$, can be forged. In this paper, we incorporate and extend Yee's work. We define a set of security properties that are more general and thus address a conspiracy of servers. Our protocols prevent a shop from being able to modify offers collected later by the agent, including its own offer. There are also fewer possibilities to truncate a chain of collected offers without being detectable by the agent originator. The comparison shopping application is used throughout the paper. In the rest of this paper we omit the directory and simply assume that an agent finds its way to the shops of interest.

The paper is organized as follows. In Section 2, we describe our trust model, give the definitions and notation used in the paper, and formalize our goals. Section 3 describes two protocols usable in an environment where a public-key infrastructure exists. In Section 4, we describe two protocols that assume only that the originator's public key is known. Finally, conclusions are drawn in Section 5.

2 Trust Model

The purpose of the system is to let agents travel from server to server, collecting intermediate results. We do not care how much work had been done on a server prior to finding the result and whether the computation ran correctly as long as the computation is independent of any input from previous servers. Comparison shopping is an example application, as the intermediate result, the price, being independent of previous price quotes.[1] The primary goal, however, is to guaran-

[1] Offers of previous shops are kept confidential in order not to influence future price quotes. We do not consider the possibility that a server may learn about the prices of competitors by querying them directly.

tee that the partial results collected by the agent and returned to the originator are not undetectably tampered with.

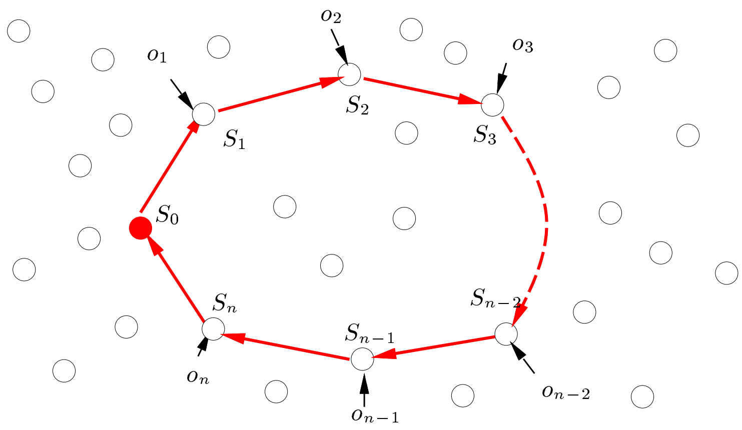

Fig. 1. Agent itinerary to collect offers o_1 to o_n

In the case of a fixed itinerary, i.e. at agent departure all servers are known as well as the order in which the agent will visit them, there are mechanisms to ensure that all intermediate results will be received and no result will have been tampered with. However, the following classes of itineraries make it more difficult to preserve the integrity of the intermediate results:

– the nodes to be visited are predetermined but the agent is free to select the next node;
– all nodes are known but the agent is free to decide which m out of n nodes to visit and in what order;
– the agent might visit nodes not known at departure.

In this paper we address in particular the latter two classes; the itinerary of the agent is not (completely) fixed when the agent leaves the originator's site. This means that any place that will host the agent can determine the next hop or hops of the agent. The actual decision process is beyond the scope of this paper.

In a comparison shopping application, each server represents a shop S_i and the intermediate results are the offers o_i of the shops returned to the originator S_0. We assume that offers contain the name of the originator. The itinerary of an agent is shown in Figure 1. Table 1 summarizes the notation we use.

Our protocols rely on certain cryptographic primitives. We assume that, given the signature $\mathcal{SIG}_i(m)$, anyone can extract m. In addition, the identity of signer S_i can be deduced by examining the signature $\mathcal{SIG}_i(m)$. Encryption is probabilistic and secure against chosen plaintext attacks. The one-way hash function

Table 1. Model Notation

$S_0 = S_{n+1}$	Originator.
S_i, $1 \leq i \leq n$	Shops.
o_0	A token issued by S_0 to identify the agent instance on return.
o_i, $1 \leq i \leq n$	Offer from S_i.
O_i, $1 \leq i \leq n$	Encapsulated offer (o_i with protective encapsulation) from S_i.
h_i, $1 \leq i \leq n$	Integrity check value associated with O_i
$O_0, O_1, O_2, \ldots, O_n$	The chain of encapsulated offers from shops $S_1, S_2, \ldots, S_n$.

used shall leak no information. Table 2 summarizes the cryptographic notation we use.

Table 2. Cryptography Notation

r_i	A nonce generated by S_i.
$\mathcal{ENC}_0(m)$	Message m encrypted with the public encryption key of S_0.
$\mathcal{SIG}_i(m)$	Signature of S_i on a message m.
$\mathcal{H}(m)$	A one-way, collision-free hash function (e.g., SHA-1[4]).
$\mathcal{MAC}_k(m)$	Message Authentication Code generated with secret k.
$[m]$	A message m sent via a confidential channel.
$Alice \rightarrow Bob$: m	Alice sending message m to Bob.

A chain $O_0, O_1, O_2, \ldots, O_n$ is an ordered sequence of encapsulated offers such that each entry of the chain depends on the previous and/or the next members. This dependency is specified by a *chaining relation*. We say that a chain is *valid at* O_k if the chaining relation holds at each link up to and including O_k. A chain is *valid* if the chain is valid at its last element. If defined properly, any modification of the chain, including deletion or insertion of elements, will invalidate the link for all members dependent on the altered element.

If two or more malicious servers conspire, they might control all data established between the first and the last conspirator on the itinerary. The same situation arises when an agent visits the same shop twice. We say that two servers conspire if they exchange secrets used in building the chaining relation.

2.1 Attacks

After capturing an agent holding a chain of offers $O_0, O_1, \ldots, O_m$, the attacker might (1) *modify* existing encapsulated offers by changing the purported sender S_i or by altering the offer o_i for any i, $1 \leq i \leq m$; (2) *insert* offers that would not have been included without the intervention of the attacker; or (3) *delete* any of the existing offers. The particular attack where all offers after the ith offer are deleted is called *truncation* at i. The combined attack where the attacker simultaneously truncates at i and appends fake offers is referred to as *growing a fake stem* at i.

As an agent might visit the same shop twice or as shops might conspire, a malicious shop might modify an agent before, at, or after its own node. This might lead to a stem (add), truncation (delete), or other misrouted itinerary.

2.2 Security Properties

Below we define a number of properties that are defined with respect to an attacker that captured a mobile agent holding a chain of encapsulated offers $O_0, O_1, \ldots, O_m$. Thus, the agent has visited an undetermined number m of shops, $m \leq n$; some shops, but not shop S_m, may conspire with the attacker, possibly being shop S_{m+1}. Let i range over $1, \ldots, m$.

- *Data Confidentiality:* Only the originator can extract the offer o_i of shop S_i.
- *Non-repudiability:* Shop S_i cannot repudiate its offer o_i once it has been received by originator S_0.
- *Forward Privacy:* None of the identities of the creator of offer o_i can be extracted.
- *Strong Forward Integrity:* None of the encapsulated offers O_k, where $k < m$, can be modified.
- *Publicly Verifiable Forward Integrity:* Anyone can verify the offer o_i by checking whether the chain is valid at O_i. Thus, any intermediate server or the agent itself can verify the partial results already collected by the agent and abort the agent's itinerary if necessary.
- *Insertion resilience:* No offer can be inserted at i unless explicitly allowed; i.e. the attacker is actually shop S_{m+1}.
- *Truncation Resilience:* The chain can only be truncated at i if shop S_i colludes with the attacker.

By relaxing Yee's assumption of the "honest prefix" of the agent's route, our definition ensures that it is impossible to forge an offer, even when it was made by oneself or by a colluding server. Therefore, we will also refer to Yee's definition as weak forward integrity. Correspondingly, we speak of weak truncation resilience when a chain can be truncated at i if shop S_j, where $j < i$, colludes with the attacker.

Note that forward integrity is defined with respect to O_m, assuming that O_m cannot be changed or removed. Therefore, strong forward integrity does not imply resistance against a truncation attack at O_k with $k < m$. Moreover, truncation resilience does not preclude a denial-of-service attack in which all offers are removed (i.e. $k = 0$), but it would be detected by originator S_0.

3 Per-server Digital Signatures

We assume that a public-key infrastructure exists. Each entity, be it the originator or a shop, has a distinguished name and a public signature verification key bound to the name via a certificate issued by a certification authority. There is a directory service from which certificates can be retrieved.

For clarification and reference, we formalize Yee's original per-server digital signature protocol [9] where each shop signs its offer using its own key, so the result is non-repudiable and unforgeable. Optionally, the offer may be encrypted by the originator's public key to ensure data confidentiality.

- *Encapsulated Offer*
  - $O_i = \mathcal{SIG}_i(\mathcal{ENC}_0(o_i)),\ 0 \le i \le n$
- *Protocol*
  - $S_i \to S_{i+1}\colon \{O_k \mid 1 \le k \le i\},\ 0 \le i \le n$

However, to detect the removal of offers, Yee's protocol requires that the agent originator knows at least the length of the agent's itinerary. Furthermore, if a server catches the agent again later, it is able to modify its own previous result.

The two protocols presented next provide strong forward integrity and truncation resilience. Protocol P1 allows anyone to authenticate the shops involved, whereas protocol P2 hides the identity of the involved shops to everybody except the originator.

3.1 Publicly Verifiable Chained Digital Signature Protocol (P1)

Protocol P1 extends Yee's per-server digital signature protocol [9] with a hash chain that links the offer of the previous shop with the identity of the next shop. This disables a shop from modifying its own offer later, and thus strong forward integrity is achieved.

Description

- *Encapsulated Offer*:
  - $O_i = \mathcal{SIG}_i(\mathcal{ENC}_0(o_i, r_i), h_i),\ 0 \le i \le n$
- *Chaining Relation*
  - $h_0 = \mathcal{H}(r_0, S_1)$
  - $h_i = \mathcal{H}(O_{i-1}, S_{i+1}),\ 1 \le i \le n$
- *Protocol*
  - $S_i \to S_{i+1}\colon \{O_k \mid 0 \le k \le i\},\ 0 \le i \le n$

To begin the protocol, the originator S_0 picks a random number r_0. Next, it computes the hash value h_0, the anchor value of the chaining relation, by applying $\mathcal{H}()$ over r_0 and the identity of the first shop S_1. Then it encrypts the token o_0 and the random value r_0 with its own public key. Finally, it constructs a dummy encapsulated offer O_0 by signing the encryption and h_0, which is sent to the first shop S_1.

When the agent arrives at shop S_i, it contains the set of previously collected encapsulated offers, including encapsulated offer O_{i-1} from which the next hash value h_i can be computed. In general, an encapsulated offer o_i $(i > 0)$ contains the following two items:

– the offer o_i probabilistically encrypted so that only S_0 can retrieve it, and
– a hash of the previous offer O_{i-1} concatenated with the identity of the next shop S_{i+1}.

The second item serves two purposes. First, it links the previous offer with the current offer: O_{i-1} cannot be modified without modifying O_i as well. In fact, even shop S_{i-1} cannot modify its own offer later without invalidating the chain at O_i. Second, the inclusion of the identity of the next shop guarantees that no one other than S_{i+1} can append the next offer. Figure 2 illustrates the protocol, where the dashed lines show the linkage of information.

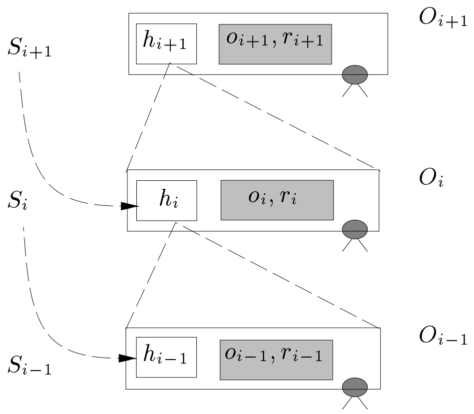

Fig. 2. Chaining in P1

Goals

Data Confidentiality. If the encryption scheme is secure, no one other than S_0 can decrypt $\mathcal{ENC}_0(o_i, r_i)$ to extract o_i. As the encryption is probabilistic, no one other than S_0 and S_i will get *any*[2] information about o_i given $\mathcal{ENC}_0(o_i, r_0)$.

Non-repudiability. The encapsulated offer O_i contains a signature by shop S_i. If the signature scheme is secure, S_i cannot repudiate O_i.

[2] We assume that the encryption scheme is secure against chosen plaintext attacks. We also include the random factor r_i in the encryption. Both together imply that even when an attacker knows one or more encryptions of a given plaintext, he has no better chance of recognizing yet another encryption of the *same* plaintext than he would have had without those encryptions.

Strong Forward Integrity. Suppose the attacker leaves O_m intact but changes the k^{th} $(k < m)$ offer. Assume that $k = m-1$ and that the attacker replaced O_{m-1} by O'_{m-1}. As O_m was not changed, the chaining relationship at m must still hold. O_m is of the form $\mathcal{SIG}_m(\mathcal{ENC}_0(o_m, r_m), h_m)$, where $h_m = \mathcal{H}(O_{m-1}, S_{m+1})$. In order for the chaining relation to hold, $\mathcal{H}(O'_{m-1}, S_{m+1}) = \mathcal{H}(O_{m-1}, S_{m+1})$. But this violates our assumption that the hash function $\mathcal{H}()$ is collision-free. Hence as long as $\mathcal{H}()$ is collision-free, it is not possible to modify O_{m-1} without modifying O_m and still maintain the chaining relation at O_m. Similarly, it is not possible to modify any O_k, $k < m-1$ without modifying O_{k+1} and still maintain the chaining relation at O_{k+1}. The property follows by induction.

Publicly Verifiable Forward Integrity. Consider someone examining the chain $O_0, O_1, \ldots, O_i$. Each O_k will appear to be $O_k = \mathcal{SIG}_k(X_k, h_k)$, where only X_k is an uninterpretable bit string. Thus the examiner can verify whether the chaining relation holds at each link and whether the signature on each O_i is valid.

Insertion Resilience. We follow the same line of reasoning as above. Given a chain of encapsulated offers $O_0, O_1, \ldots, O_m$, no one can insert an offer anywhere between O_1 and O_m because the chaining relation will not be respected after the insertion point. However, S_{m+1} can insert an offer after S_m because it is explicitly allowed to do so by the protocol.

Truncation Resilience. Given a chain of encapsulated offers $O_0, O_1, \ldots, O_m$, any truncation at k with $k < m$ will be detected unless the attacker is able to insert simultaneously an offer in the name of S_{k+1}. Thus, the truncation resilience property is fulfilled. However, because any shop in the chain can truncate all offers following its own, the protocol allows the attacker to insert other fake offers before the agent is sent back to the originator.

3.2 Chained Digital Signature Protocol with Forward Privacy (P2)

By changing the order of signing and encrypting an offer, it is possible to hide the identity of the shops that provided offers.

Description

- *Encapsulated Offer*:
  - $O_i = \mathcal{ENC}_0(\mathcal{SIG}_i(o_i), r_i), h_i,\ 0 \leq i \leq n$
- *Chaining Relation*
  - $h_0 = \mathcal{H}(r_0, S_1)$
  - $h_i = \mathcal{H}(O_{i-1}, r_i, S_{i+1}),\ 1 \leq i \leq n$
- *Protocol*
  - $S_i \rightarrow S_{i+1}$: $\{O_k \mid 0 \leq k \leq i\}$

Thus, the shop signature is encrypted and can only be revealed by the originator.

Goals

Instead of publicly verifiable forward integrity, protocol P2 achieves forward privacy. The other properties of protocol P1 are still satisfied. The proofs in Section 3.1 apply to these cases with no modifications.

Forward Privacy. If the encryption scheme is secure, no one other than S_0 can decrypt a sealed offer O_i. As the components of O_i (encryption and hashing) are randomized, they will leak no information about their contents, and will resist dictionary attacks as well. Thus, no one other than originator S_0 can determine the creator of the sealed offer.

4 Hash Chains

Now we assume that there is no public-key infrastructure. However, each shop knows the public key of the originator. Let us first examine why this assumption is realistic. An example scenario is when a big company solicits bids from small subcontractors to carry out a project (invitation of tenders). All subcontractors know the public key of the big company. There is no need for a non-repudiable contract offer. The big company can examine all the offers collected by the agent, pick the offer(s) of interest, and contact the subcontractors directly.

Yee's Partial Result Authentication Code demonstrates the authenticity of an intermediate agent state or partial result that resulted from having run on a server [9]. With each server, there is an associated secret key used to compute an authentication tag on this data. After a tag is computed, the agent/server takes care to erase the secret key associated with the current server prior to migrating to the next server. For efficiency, Yee uses a secure hash function to generate the secret keys from a single secret h_0.

- *Encapsulated Offer*
  - $O_i = o_i, \mathcal{MAC}_{h_i}(o_i),\ 0 \le i \le n$
- *Chaining Relation*
  - $h_{i+1} = \mathcal{H}(h_i),\ 1 \le i \le n$
- *Protocol*
  - $S_i \to S_{i+1}\colon \{O_j \mid 0 \le j \le i\}, [h_{i+1}],\ 0 \le i \le n$

The erasure step ensures weak forward integrity. The protocol also satifies insertion and weak truncation resilience. If selected, data confidentiality and forward privacy can be realized.

4.1 Chained MAC Protocol (P3)

Protocol P3 extends Yee's message authentication code protocol. Each shop S_i generates a random number r_i and encrypts it together with the offer and the identifier of the next shop under the originator's public key. The value r_i, being part of the encapsulated offer, is an input to the computation of the next hash key h_{i+1} and thus prevents shops from modifying their own offers later. More importantly, it prevents a shop S_i from being able to modify the offers of shop S_j, where $j > i$.

The protocol requires that each pair of shops be connected via a confidential channel.

Description

- *Encapsulated Offer*:
  - $O_i = \mathcal{ENC}_0(r_i, o_i, S_{i+1}),\ 0 \leq i \leq n$
- *Chaining Relation*:
  - $h_1 = \mathcal{ENC}_0(r_0, o_0, S_1)$
  - $h_{i+1} = \mathcal{H}(h_i, r_i, o_i, S_{i+1}),\ 1 \leq i \leq n$
- *Protocol*:
  - $S_i \rightarrow S_{i+1}$: $\{O_k \mid 0 \leq k \leq i\}, [h_{i+1}],\ 0 \leq i \leq n$

Figure 3 depicts the chaining procedure for this protocol. The term h_i serves to link all the previous encapsulated offers to the current offer. The term r_i prevents a shop $S_k (k < i)$ from being able to replace any encapsulated offer O_j that comes *after* O_i $(j > i)$ in the chain because S_k does not know (and cannot predict) the chaining factor h_j that must be embedded into O_j.

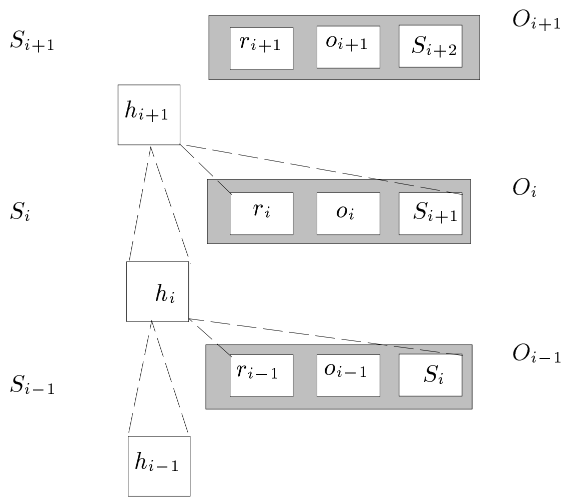

Fig. 3. Chaining in P3

Embedding of S_{i+1} provides no authentication as in the previous protocols. It can merely help S_0 make a sanity check. If h_i is embedded in the encrypted offer O_i, the originator can determine at which shop the chaining relation broke.

The proof for insertion and truncation resilience is identical to that in Section 3.1.

Data Confidentiality. If the encryption scheme is secure, no one other than originator S_0 can decrypt $O_i = \mathcal{ENC}_0(r_i, o_i, S_{i+1})$ to extract offer o_i. The random component r_i renders the encryption probabilistic. Therefore, given O_i, no one other than S_0 can infer any information about o_i.

Strong Forward Integrity. Assume that an attacker can leave O_m, h_{m+1} (i.e., the quantities added to the state of the agent by S_m) intact and still change O_k, $0 \leq k < m$ without being detected by S_0. Assume that $k = m - 1$ and that the attacker replaced O_{m-1} by O'_{m-1}. We can assume that O'_{m-1} is of the form $\mathcal{ENC}_0(r'_i, o'_i, S_{i+1})$, otherwise S_0 will reject the chain as malformed, thereby violating our assumption. As h_{m+1} and O_m were not changed, the chaining relationship at m must still hold. Suppose $\mathcal{H}(h'_{m-1}, r'_{m-1}, o'_{m-1}, S_m)$ evaluates to h'_m. As the chaining relationship at O_m must be preserved, it must be the case that $\mathcal{H}(h'_m, r_m, o_m, S_{m+1}) = h_{m+1}$. But the attacker (who is by definition not the same as S_m) does not know r_m and o_m. Therefore, the only way he can satisfy the above relationship is to make sure that $h'_m = h_m$. This implies that $\mathcal{H}(h'_{m-1}, r'_{m-1}, o'_{m-1}, S_m) = \mathcal{H}(h_{m-1}, r_{m-1}, o_{m-1}, S_m)$. But this violates our assumption that the hash function $\mathcal{H}()$ is collision-free. Hence as long as $\mathcal{H}()$ is collision-free, it is not possible to modify O_{m-1} without modifying O_m, h_{m+1} and still maintain the chaining relation at O_m. Similarly, it is not possible to modify any $O_k, k < m - 1$ without modifying O_{k+1}, h_{k+1} and still maintain the chaining relation at O_{k+1}. The property follows by induction.

Forward Privacy. If the encryption scheme is secure, no one other than S_0 can decrypt a given sealed offer O_i. The random salt r_i makes O_i a randomized encryption, thereby ensuring that it will leak no information about its contents. Thus, no one other than originator S_0 can determine the creator of the given sealed offer.

4.2 Publicly Verifiable Chained Signatures (P4)

Yee also introduces a scheme that makes PRACs publicly verifiable, i.e. any intermediate server not sharing the secret key with the originator can itself verify the partial results already collected by the agent. For this purpose, each server provides its successor with a secret signature function and certifies the corresponding verification predicate. With the received signature/verification function pair, a server signs its partial result and certifies a new verification function to be used by the next server. The originator provides the agent with the initial secret signature function and a certified verification predicate. Each server destroys the signature function received but adds the new verification function generated itself to the list of certified verification predicates carried by the agent.

With the above scheme, the originator must not precalculate the signature function/verification pairs, and the list of certified verification functions grows linearly with the number of shops visited. One-time digital signatures as well as public-key digital signatures can be used to implement the digital signature system. Let $\mathcal{SIG}_{\overline{y_i}}(m)$ denote the (one-time) signature of shop S_i on a message m generated with the secret key $\overline{y_i}$.

Protocol P4 uses only the public key of the originator to set up the signature chain. The originator includes the initial public key of the signature chain in the encapsulated offer O_0 signed with its public key-based digital signature. The corresponding secret key $\overline{y_1}$ is then sent together with the encapsulated offer to shop S_1. Each shop S_i generates a secret/public key pair $\overline{y_{i+1}}/y_{i+1}$ and includes the public key y_{i+1} in the encapsulated offer O_i. The encapsulated offer is signed with the secret key $\overline{y_i}$ received from the previous shop.

Description

- *Encapsulated Offer*:
  - $O_0 = \mathcal{SIG}_0(\mathcal{ENC}_0(o_0, r_0), h_0, y_1)$
  - $O_i = \mathcal{SIG}_{\overline{y_i}}(\mathcal{ENC}_0(o_i, r_i), h_i, y_{i+1}),\ 1 \le i \le n$
- *Chaining Relation*
  - $h_0 = \mathcal{H}(r_0, S_1)$
  - $h_i = \mathcal{H}(O_{i-1}, S_{i+1}),\ 1 \le i \le n$
- *Protocol*
  - $S_i \to S_{i+1}\colon \{O_k \mid 0 \le k \le i\}, [\overline{y_{i+1}}],\ 0 \le i \le n$

Applying above signature chain to protocol P2, it is even feasible to achieve forward privacy together with publicly verifiable forward integrity.

5 Conclusion

We developed a number of protocols that protect the results established by free-roaming mobile agents. All protocols, even if relying only on a secret hash chain and the public key of the originator, achieve strong forward integrity, truncation resilience, and data confidentiality. In addition, publicly verifiable forward integrity and forward privacy can be achieved. With a public-key infrastructure in place, non-repudiability of offers is added. By having disabled a shop from modifying even its own offer later, our protocols tolerate collusion among shops. Application areas of these protocols include comparison shopping, bidding, and network routing [2].

Our definition of forward integrity has the constraint that the attacker does not change the last element O_m in the chain of encapsulated offers. Lifting this restriction, only protocols where the originator can authenticate shop offers still satisfy the strengthened definition. Otherwise, the attacker has always the possibility to truncate the chain and to grow a fake stem.

Truncation is the most difficult problem to tackle in preserving the state of the offer chains. To achieve even stronger resistance against truncation, we see the following three possibilities:

1. Whenever an agent obtains addresses of new shops to be visited, these names could be stored in the agent, for example by embedding the partially precomputed itinerary $S_{i+1}, S_{i+2}, \ldots, S_{i+k}$ into O_i. Thus it would be possible to check after returning home whether the agent had been at all shops listed. Of course, this implies that a shop must also express the fact that it does not desire or is not able to give an offer.
2. Originator S_0 broadcasts or publishes an itinerary after the fact, allowing victim shops to complain [6].
3. If the forward or next-hop function is verifiable, the originator could recompute the itinerary.

If shops were anonymous, it would be more difficult for them to conspire. For example, assume that the agent first visits a trader that provides it with a list of anonymous shops. Whenever the agent moves to another shop, it reshuffles the collected data to make it impossible to track the agent. Even conspiring shops would not be able to identify each other or the moving agent.

Acknowledgments

The authors thank Michael Steiner, Gene Tsudik, and Michael Waidner for their many helpful suggestions.

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*, pages 75–90, 1996.
[2] G. Di Caro and M. Dorigo. Mobile agents for adaptive routing. In *31st Hawaii International Conference on System Science (HICSS-31)*, Big Island of Hawaii, January 6–9,1998.
[3] B. Doorenbos, O. Etzioni, and D. Weld. A scalable comparison-shopping agent for the world-wide web. In *AGENTS-97*, 1997.
[4] NIST National Institute of Standards and Technology (Computer Systems Laboratory). Secure hash standard. Federal Information Processing Standards Publication FIPS PUB 180-1, April 1995.
[5] D. Rus, R. Gray, and D. Kotz. Transportable information agents. In M. Huhns and M. Singh, editors, *Readings in Agents*. Morgan Kaufmann Publishers, 1997.
[6] G. Tsudik. Personal communication.
[7] G. Vigna. Protecting mobile agents through tracing. In *Third Workshop on Mobile Object Systems*, June 1997.
[8] J.E. White. Mobile agents. In J. Bradshaw, editor, *Software Agents*. AAAI Press and MIT Press, 1996.
[9] B.S. Yee. A sanctuary for mobile agents. Technical Report CS97-537, UC San Diego, Department of Computer Science and Engineering, April 1997.