Abstract

The Internet is continuously extending to mobile devices. This implies that applications should be tailored to make efficient use of the limited resources of such devices. Mobile agents are a promising solution for this purpose. However, secure protocols are required in order to assure confidentiality and integrity of the data such an agent collects. This paper describes 6 such security protocols presented in the scientific literature. These protocols are analyzed with respect to a series of security requirements that should be satisfied by such protocols. An overview of some published attacks on these protocols and the proposed solutions is also presented.

1 Introduction

Nowadays, virtual transactions on the Internet have become a common practice. Throughout the world people are performing activities that imply some form of secure communication with other parties, and possibly also electronic payment. The classical scenario involves a multi-step communication between a user running some client software on her local host and a remote server. This communication can prove quite time consuming for the user. Consider the frequent situation where a user searches for the online shop offering the lowest price of a product. This may require surfing several different websites and comparing prices. Other examples like electronic auctions require may require a high level of interaction.

Generally, Internet users try to spend less time doing business and more time on procrastination. This is a reason why an increasing number of companies try to offer clients what they need as quickly and using as few interactions as possible. Moreover, the current trend is towards extending services offered on desktop PCs to mobile devices. These devices have a low amount of processing and power resources, which limit their capabilities. Also their connection to mobile Internet is expensive and offers low bandwidths. Therefore, users of such mobile devices prefer to generate as little traffic as possible.

A solution meant to resolve the aforementioned problems is expressed in the form of mobile agents. Such a mobile agent is represented by an independent piece of software that can operate autonomously in order to fulfill a given task. The agent is assigned a task by an originating host, after which it is sent to a series of other hosts. Mobile agents travel from one host to another where they are locally executed, gathering information for the originator. Such agents are often intelligent, and may enabled with the so called self-CHOP attributes proposed by IBM [4]. After finishing their task, mobile agents return back to the originator to report the collected results. Therefore, the intervention of the originator is only required before sending and upon receiving the agent.

From the previous description of mobile agents, several security concerns arise. These are presented in depth by Claessens et al. [3], we only briefly mention them here. An agent may need to carry sensitive information of the originator which should not be seen by other hosts. Furthermore, the originating user that sends the agent may be malicious. This poses a security threat to honest service providers that receive and execute a potential malicious agent. Moreover, such a malicious user could also deny having sent the agent at all.

This paper investigates the security of protocols used by mobile agents. Section 2 describes the security challenges associated with mobile agents. Six security protocols presented in the scientific literature are described in Sections 3 and 4. Attacks on some of these protocols are presented in Section 5 and solutions to prevent these attacks are described in section 6. Finally, Section 7 presents some conclusions.
2 Problem Analysis

The example of comparison shopping that was briefly mentioned in the introduction will be used throughout this paper in order to exemplify the protocols involving mobile agents, and their security properties. The task of a mobile agent programmed to do comparison shopping is to travel from server to server, collecting offers for goods matching the terms set by the originating user. This task needs to be done in such a manner as to prohibit any attacker (including the electronic merchants) from reading or tampering with any of the offers collected from previously visited stores. Hence, the computation of the offer done by each visited host needs to be independent of all previous offers.

An ordered list of the hosts, showing the sequence of shops visited by the mobile agent is called an itinerary. There exist several possibilities for creating an agent itinerary. The least flexible solution is to have a fixed itinerary set by the originator. In such a case, the integrity of the results returned by the agent after it finished collecting offers, can be easily verified.

In contrast, there exist other classes of itineraries, which provide more flexibility. Agents that follow such a flexible itinerary are also called free-roaming agents. For instance the set of visited shops may be fixed by the originator, but the agent may freely chose the order in which the shops will be visited. The agent may also choose not to visit some of the shops given by the originator (e.g. the shop may be offline). A higher degree of flexibility may be employed by allowing the agent to visit shops which were not given by the originator. However, a higher degree of flexibility for the itinerary also implies more possibilities of attacking the mobile agent.

Security protocols for mobile agents having fixed or flexible itineraries will be described in Section 3. In order to describe these protocols we will employ the same notation used by Karjoth et al. [6]. The comparison shopping application is modeled using the notation presented in Table 1. The protocols which will be described later, depend on the cryptographic primitives described in Table 2.

Each entry of the chain of encapsulated offers \((O_0, O_1, ..., O_n)\) may depend on the previous and/or following hosts that the mobile agent visits. In [6] this dependency is specified by a so called chaining relation. The chain is said to be valid at \(O_i\) if the chaining relation is satisfied for each link starting from \(O_0\) up to and including \(O_i\). Therefore, a chain is said to be valid, if it is valid at its last element. The main challenge in defining a chaining relation is to prevent attackers from tampering with any elements of the chain, without invalidating the chain.

The chain of offers of an agent can be basically tampered with in three ways:

- modification of one or more senders or of their offers;
- insertion of new offers at an arbitrary point in the chain;
- deletion of one or more existing offers.

In [6] the particular attack of deleting all offers after the \(i\)-th one, is called a truncation at \(i\). Also, an attack comprising of a truncation at \(i\) and then an insertion of fake offers at the end of a chain, is called growing a fake stem at \(i\).

When two or more malicious hosts exchange the secrets used in building the chaining relation, it is said that those servers conspire. In such a case, the attackers may be able to change offers contained in the subchain between any two conspiring server offers, without invalidating the chain. More attacks against mobile agents will be described in Section 5.

Karjoth et al. [6] define a set of desirable security properties for mobile agent protocols. Assuming that an agent having a chain \((O_0, O_1, ..., O_m, m \leq n)\) arrives at a malicious host \(S_{m+1}\), the following properties should hold:

\[
\begin{align*}
S_0 &= S_{n+1} & \text{The first and last nodes representing the originator} \\
S_i, 1 \leq i \leq n & \text{Shops the agent visits} \\
o_0 & \text{Token issued by the originator, used to identify the agent instance upon its return} \\
o_i, 1 \leq i \leq n & \text{Offer from shop \(i\)} \\
o_i & \text{with protective encapsulation} \\
h_i, 1 \leq i \leq n & \text{Integrity check value associated with \(O_i\)} \\
O_0, O_1, ..., O_n & \text{Chain of encapsulated offers}
\end{align*}
\]

Table 1: Model Notation from [6]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r_i)</td>
<td>A nonce generated by (S_i)</td>
</tr>
<tr>
<td>(\text{ENC}_0(m))</td>
<td>Encryption of message (m) using the public encryption key of (S_0)</td>
</tr>
<tr>
<td>(\text{SIG}_i(m))</td>
<td>Signature of (S_i) on a message (m)</td>
</tr>
<tr>
<td>(H(m))</td>
<td>A one-way, collision-free hash function</td>
</tr>
<tr>
<td>(\text{MAC}_k(m))</td>
<td>Message Authentication Code generated using secret key (k)</td>
</tr>
<tr>
<td>([m])</td>
<td>Message (m) sent via a confidential channel</td>
</tr>
<tr>
<td>(A \rightarrow B : m)</td>
<td>(A) sends message (m) to (B)</td>
</tr>
</tbody>
</table>

Table 2: Cryptography Notation from [6]
- **Data confidentiality**: the originator is the single entity that can read any offer \( o_i \) from the chain;

- **Non-repudiability**: no host \( S_i \) may deny having made the offer \( o_i \), after the chain is received by the originator;

- **Forward privacy**: none of the identities of the visited hosts, can be extracted by someone other than the originating user;

- **Strong forward integrity**: if any encapsulated offer \( O_i \) is tampered with, it will invalidate the chain; this does not imply resistance against truncation attacks at \( O_k, k < m \);

- **Publicly verifiable forward integrity**: any entity can verify the validity of the chain at any encapsulated offer \( O_i \);

- **Insertion resilience**: no offer may be inserted in arbitrary points of the chain; a host may insert an offer at the end of the chain only if it is the host which the agent was supposed to visit at that point in its itinerary;

- **Truncation resilience**: the current chain can only be truncated at \( i \) if host \( S_i \) colludes with the attacker; this does not eliminate denial-of-service attacks (e.g. all offers are removed).

An important property which was not mentioned in the previous enumeration is **unforgeability of encapsulated offers**: no entity should be able to impersonate a host from the agent’s itinerary and post a fake offer in that host’s name. This is a complementary property to non-repudiability, because if an attacker impersonates a shop giving an extremely attractive offer, employing only non-repudiability would probably cause losses to the merchant. Therefore, the unforgeability property is meant to protect the hosts which the agent visits. This property is not given in [6], even though the paper presents protocols that satisfy it. This property has a certain connection to the insertion resilience property described by Karjoth, however in our opinion the latter property would be best defined as done by De Weger [5] (Section 5.3.3). Consequently, the unforgeability property would naturally fit in the previous list of security properties.

### 3 Publicly verifiable digital signature protocols

In this section we assume that there exists a public-key infrastructure (PKI), i.e. each participant of the system can be uniquely identified by a name-key pair via a certificate issued by a Certification Authority (CA). One of the first mobile agent protocols using publicly verifiable digital signatures was described by Yee in a Technical Report; this later became a publication [10]. In this protocol, each shop may first encrypt its offer using the originator’s public key, after which the resulting ciphertext is signed using the signing key of that shop. Karjoth et al. formalize Yee’s protocol in the following way:

\[
O_i = \text{SIG}_i(\text{ENC}_O(o_i))
\]

By encrypting the offer using the originators public key, data-confidentiality is satisfied. The signature of the encrypted offer assures non-repudiability and unforgeability. However, forward privacy is not satisfied since any entity in the system can obtain the identities of the shops which have added offers to the “chain”. This can be done because anyone can see who signed the encapsulated offer \( O_i \).

Strong forward integrity and insertion resilience are not satisfied because a shop can always change its offer or insert a new offer in any position of the “chain”, before the agent returns to the originator. This happens because all the encapsulated offers are independent of each other, hence in this protocol the sequence \( O_0, O_1, \ldots, O_n \), is not a chain in the strict sense, i.e. as defined by Karjoth et al. Due to this reason publicly verifiable forward integrity cannot hold, because the integrity of the sequence of offers is not satisfied. In the context of flexible itineraries, the truncation resilience is not satisfied because any attacker may truncate any set of offers. Moreover, the attacker may even delete any offers from arbitrary positions. However, for fixed itineraries, Yee’s protocol satisfies both insertion and truncation resilience.

Moreover, all the protocols described in this paper satisfy these two resilience properties for fixed itineraries.

#### 3.1 Publicly verifiable chained digital signature protocol (P1)

The first protocol described by Karjoth et al. [6] extends the previously described protocol of
Yee. Using a chaining relation, the encapsulated offer of the current shop, links the encapsulated offer of the previous shop with the identity of the next shop. Hence each encapsulated offer links three consecutive entities in the itinerary. The protocol is formally described as:

\[ h_0 = H(r_0, S_1) \]
\[ h_i = H(O_{i-1}, S_{i+1}), 1 \leq i \leq n \]
\[ O_i = SIG_i(ENC_0(o_i, r_i), h_i), 0 \leq i \leq n \]
\[ S_i \rightarrow S_{i+1} : \{ O_k \mid 0 \leq k \leq i \}, 0 \leq i \leq n \]

The first line of the previous protocol is executed only by the initiator. It uses a random nonce instead of an encapsulated offer for the first input of the hash function, because at that point in the protocol there does not exist an encapsulated offer of a previous host. The initiator then constructs a dummy encapsulated offer \( O_0 \) which is sent to the first shop \( S_1 \). Afterwards, on every other host the agent visits steps 2, 3 and 4 of the previous protocol are executed.

Data confidentiality is boosted with respect to the protocol described by Yee, by using non-deterministic encryption, i.e. encrypting the offer along with a random nonce. Using this approach ensures strong secrecy of the offers of the visited shops. Non-repudiability and unforgeability are both satisfied because the encapsulated offer \( O_i \), contains a signature by shop \( S_i \). Since any entity can check who this signature belongs to, forward privacy is violated.

Strong forward integrity is satisfied by this protocol, because of the chaining relation linking the encapsulated offers. Suppose an attacker replaced an arbitrary encapsulated offer \( O_{i-1} \) by \( O_{i-1} \). This would cause the value \( h_i \) stored in the encapsulated offer \( O_i \) to be invalidated, because \( H(O_{i-1}, S_{i+1}) \neq H(O_{i-1}, S_{i+1}) \). This relation is true under the assumption of a collision-free hash function \( H \). Therefore modifying any \( O_{i-1} \) would also imply modifying \( O_i \), causing a ripple effect towards the end of the chain. In other words, modifying an encapsulated offer \( O_{i-1} \) would also require modifying every encapsulated offer after it. This argumentation can also be extended to proving why insertion resilience does not hold. Therefore, if an attacker inserts an encapsulated offer \( O_i \) at an arbitrary point \( i \) in the chain, then all of the offers following it would need to be changed as well. Moreover, if the identify of the host which inserted the offer is not the same as the one that is now following it (\( O_i \)), the previous encapsulated offer \( (O_{i-1}) \) would also need to be changed because:

\[ H(O_{i-2}, S_i) \neq H(O_{i-2}, S_i) \]

Publicly verifiable forward integrity is satisfied because any entity can verify the validity of the chain \( O_0, O_1, ..., O_i, \) up to the last encapsulated offer. Each \( O_k \) from the previous chain contains a signed hash value (\( h_k \)). By verifying the signature one obtains \( h_k \) and also the identify of the signing host \( S_k \). Afterwards, one just needs to verify that \( h_k = H(O_{k-1}, S_{k+1}) \) for every \( k \in \{1, 2, ..., i\}, \) in order to be sure that the chain is valid.

Given any chain of encapsulated offers \( O_0, O_1, ..., O_m \), any truncation at \( k < m \) would be detected except for the case when the attacker can successfully insert an encapsulated offer \( O_{k+1} \). In the current context this means that the attacker must conspire with \( S_{k+1} \), assuming that it is the only host having the corresponding signing key. Therefore, any malicious host is able to perform a truncation of all offers which followed its own offer and also change its old offer at the same time.

### 3.2 Chained digital signature protocol with forward privacy (P2)

The second protocol described by Karjoth et al. [6] sacrifices the property of publicly verifiable forward integrity of the first protocol, in exchange for forward privacy. The main difference with respect to the previous protocol is changing the order of signing and encrypting an offer. Moreover, the hashing function is also randomized by including a random nonce as an input. This way, the identity of the shops which posted offers is strongly protected, also against dictionary attacks. The attack is formally described as:

\[ h_0 = H(r_0, S_1) \]
\[ h_i = H(O_{i-1}, r_i, S_{i+1}), 1 \leq i \leq n \]
\[ O_i = ENC_0(SIG_i(o_i, r_i), h_i), 0 \leq i \leq n \]
\[ S_i \rightarrow S_{i+1} : \{ O_k \mid 0 \leq k \leq i \}, 0 \leq i \leq n \]

Similarly to the previous protocol, data confidentiality is satisfied since only the originator is able to decrypt the signed offer made by a shop. Non-repudiability and unforgeability is also satisfied because of the digital signatures of the shops on the offers. We have already argued why forward privacy holds, and therefore publicly verifiable forward integrity cannot hold since this would require every entity to know the identities of every shop associated with an encapsulated offer in order to be able to verify the validity of the chaining relation. Note that if this information was available, the forward privacy property would no longer be satisfied and therefore the random nonce used when computing \( h_i \) would no longer be needed.
4 Partial Result Authentication Code (PRAC)-based protocols

The first protocol we describe belongs to Yee [10], who first introduced this type of protocols. Its goal is to prove the authenticity of an intermediate agent state (partial result) after being executed on a host. A PRAC is defined to be similar to a MAC [2] with the important difference that after a PRAC is computed, the agent takes care to delete the secret key before traveling to the next host. PRAC’s are cheaper to compute than digital signatures and have slightly different security properties. While digital signatures authenticate the origin of a message, PRACs demonstrate the authenticity of the message itself. Therefore non-repudiability of the sender is not satisfied.

Yee’s protocol is formalized by Karjoth et al. in the following way:

\[ h_{i+1} = H(h_i), 1 \leq i \leq n \]

\[ O_i = o_i, MAC_h(\alpha_i), 0 \leq i \leq n \]

\[ S_i \rightarrow S_{i+1} : \{ O_i | 0 \leq k \leq i \}, [h_{i+1}], 0 \leq i \leq n \]

An important aspect of this protocol is that it uses a confidential channel between each pair of hosts, in order to pass one the integrity check value \( h_{i+1} \). Depending on the context of the application, such a channel may be expensive to set-up and constitutes a disadvantage with respect to the previously presented protocols.

The secret key erasure feature of PRACs does not ensure strong forward integrity in case of malicious hosts. Such a host may store a copy of the secret key deleted by the agent and can therefore modify any subset of offers that follow and including its own. Therefore, the unforgeability of encapsulated offers is not satisfied by this protocol. From the previous attack, one can also observe that truncation resilience is not satisfied since a malicious server can truncate the chain of offers at any point.

Insertion resilience is satisfied because an attacker cannot insert offers in the chain at arbitrary positions without invalidating the hash chain. For instance inserting \( O'_i \) at position \( i \) in the chain using the correct integrity check value \( h'_i \) for that position, will invalidate the hash chain because either \( h'_i \neq H(h_{i-1}) \) or \( h_i \neq H(h'_i) \). This follows from the principle that no value can be inserted in an arbitrary point of a hash chain without invalidating it.

Data-confidentiality and forward privacy could be realized by the previous protocol by using some form of encryption. However, Yee designed this protocol for contexts where there does not exist a PKI. Publicly verifiable forward integrity is not satisfied because the value \( h_0 \) is a secret kept by the originator. Disclosure of \( h_0 \) would enable the attacker to modify, insert or delete any offer from the chain, undetected. This would happen because the entire hash chain is constructed starting from \( h_0 \). If an attacker had this value, he would be able to recompute any MAC value in the chain of offers.

4.1 Chained MAC protocol (P3)

This is the third protocol presented by Karjoth et al. [6]. It extends the PRAC-based protocol described previously but it requires that the other host making the offers, know the public encryption key of the originator. Karjoth et al. argue that this does not necessarily require a PKI because the originator may publish this key in a public location such that its integrity is guaranteed. The protocol is formally described as:

\[ h_1 = ENC_0(r_0, o_0, S_1) \]

\[ h_{i+1} = H(h_i, r_i, o_i, S_{i+1}), 1 \leq i \leq n \]

\[ O_i = ENC_0(r_i, o_i, S_{i+1}), 0 \leq i \leq n \]

\[ S_i \rightarrow S_{i+1} : \{ O_k | 0 \leq k \leq i \}, [h_{i+1}], 0 \leq i \leq n \]

In each step of this protocol, the host \( S_i \) generates a nonce \( r_i \) and encrypts it together with the offer \( o_i \) and the identifier of the next shop, using the public key of the originator. The same nonce is then used as an input to the
hash function used for computing the integrity check value $h_{i+1}$. The usage of the nonce in both the computation of the encapsulated offer and the hash function ensures that no host will be able to modify any previous offer without invalidating the hash chain. This is because the integrity check value links all previous offers in the chain. In order to prove that strong forward integrity holds, we suppose that an attacker replaced $O_i$ by $O'_i = ENC_0(r'_i, o'_i, S_{i+1})$. This also implies the replacement of $h_{i+1}$ by $h'_{i+1} = H(h'_i, r'_i, o'_i, S_{i+1})$, in order for the chain to be well formed when $S_0$ checks it at the end. However this would invalidate the chaining relation because $h_{i+1} \neq h'_{i+1}$ since we assumed a collision free hash function. Note that the attacker does not know the correct values $h_i$ and $r_i$ and therefore is forced to use $h'_i$ and $r'_i$. Even if the attacker is a malicious host $S_i$, who wants to change its own offer. Assuming that the host somehow managed to store the original values $h_i$ and $r_i$, the integrity check values $h_{i+1}$ and $h'_{i+1}$, would still differ because $a'_i \neq a_i$.

Insertion resilience holds since inserting an encapsulated offer $O'_i = ENC_0(r'_i, o'_i, S_i)$ at point $i$ (just before $O_i$) would invalidate the hash chain as well. This would happen because the integrity check value for that offer would be equal to $h_{i+1} = H(h'_i, r'_i, o'_i, S_i)$, therefore $S_i$ would now be the following host. Since the hash function is collision-free the following relation would hold: $h'_{i+1} = H(h'_i, r'_i, o'_i, S_i) \neq H(h_{i-1}, r_{i-1}, o_{i-1}, S_i) = h_i$. Therefore, the hash chain will be broken because $H(h'_{i+1}, r_i, o_i, S_{i+1}) \neq H(h_i, r_i, o_i, S_{i+1})$.

Similarly to the previous two protocols described by Karjoth et al., truncation resilience holds except for the case when a malicious host $S_{i+1}$ performs a truncation at $O_i$, and having somehow stored the value $h_{i+1}$, is able to append a value in its name.

Data confidentiality holds if we assume that the encryption scheme is secure, i.e. only the originator can decrypt any encapsulated offer. Since the encryption and the hashing are randomized. Forward privacy also holds, because the originator is the only one who can determine the identity of the shop that created an offer.

Since no signature scheme is used, the non-repudiability and unforgeability properties are not satisfied since an attacker may impersonate a legitimate host $S_i$ and easily make a false offer in $S_i$’s name with out being detected. This attack could for instance take place on a malicious host. First the host $S_m$ receives the correct integrity check value $h_{m+1}$ on the confidential channel and the chain of encapsulated offers. $S_m$ first makes an unattractive offer in its name and subverts the agent control flow in such a way that it sets the next host in the itinerary to one of its greatest competitors. However the host tricks the agent in believing that it was sent to the next host, while in reality is has never left $S_m$. The malicious host now impersonates its competitor making an extremely attractive offer that would also cause great loses to its competitor. Afterwards $S_m$ can release the agent on its normal itinerary, with a valid chain of offers.

Publicly verifiable forward integrity does not hold because this implies the knowledge of all the integrity check values $h_i$, $1 \leq i \leq n$, to be able to check this. However these values are not public since they are sent along a confidential channel, and their disclosure would break the integrity of the chain of encapsulated offers. Also because the hash function used to compute the values $h_i$ is a one-way function, a host cannot determine any previous values $h_j$, $j < i$.

### 4.2 Publicly verifiable chained signatures (P4)

The forth protocol presented by Karjoth et al. [6] also extends Yee’s idea of publicly verifiable PRACs. These types of PRACs facilitate the verification of partial results by any intermediate host. This can be achieved using a digital signature system without needing a PKI. Yee [10] suggests that the originator should generate a list of signature functions $SIG_i$ along with certificates for their corresponding verification functions $VERIF_i$. The verification functions would need to be signed by the originator’s signature function $SIG_0$, and the originator’s verification function $VERIF_0$ would be public. An agent would be provided with all of the signature and verification functions. After being executed on a host $S_i$, the partial result would be signed along with $VERIF_{i+1}$, using $SIG_i$, after which this signature function would be deleted (prior to migrating to the next host $S_{i+1}$).

Using this idea, any host can check whether the chain of encapsulated offers is valid. However, in the previous protocol, the resource consuming task of computing all the signature and verification functions is left to the originator. Karjoth et al. recommend the use of a one-time signature scheme [7] or a public-key signature scheme. In this way every host can compute a secret/public key pair $y_i/y_{i+1}$ and use them with the chosen signature scheme. Hence, the originator no longer needs to compute all the signature and verification functions. The pro-
The protocol is formally described as:

\[ h_0 = H(r_0, S_1) \]
\[ O_0 = SIG_0(ENC_0(o_0, r_0), h_0, y_1) \]
\[ h_i = H(O_i-1, S_{i+1}), 1 \leq i \leq n \]
\[ O_i = SIG_y(ENC_y(o_{y_i}, r_i), h_i, y_{i+1}), 1 \leq i \leq n \]
\[ S_i \rightarrow S_{i+1} : \{O_k | 0 \leq k \leq i\}, [y_{i+1}], 0 \leq i \leq n \]

The initiator first generates a nonce \( r_0 \) and a secret/public key pair \( y/y_1 \). It computes the first integrity check value \( h_0 \), linking the next host to be visited. Afterwards it encrypts its fake offer \( o_0 \) together with the random nonce. The ciphertext along with the integrity check value and the public key are all signed using its own secret signature function \( SIG_0 \). It then sends the encapsulated offer via a public channel and the secret key \( y \) via a confidential channel.

Each host \( S_i \) that receives the chain \( O_0, O_1, ..., O_{i-1} \), may first verify its integrity before doing any other computations. This can be done starting from verifying the first encapsulated offer \( O_0 \) with the public verification function of the originator \( VERIF_0 \). The public key \( y_1 \), that results from this verification, can then be used to verify \( O_1 \) using the verification function \( VERIF_y \), which uncovers public key \( y_2 \) and so on. Hence publicly verifiable forward integrity is satisfied by this protocol.

Data confidentiality holds because the non-deterministic encryption of the offer ensures that only the originator can extract an offer from the encapsulated offer. The signature scheme used does not authenticate a host, it only proves integrity of the message. Since any host can generate a pair of secret/public keys and impersonate another host, non-repudiability and unforgeability do not hold. However, due to this fact, forward privacy is satisfied. However, it could be easily strengthened by randomizing the hash function when computing the integrity check value, by using the random nonce like so: \( h_i = H(O_{i-1}, r_i, S_{i+1}) \).

The integrity check value associated to the offer is signed together with the encrypted offer, in the encapsulated offer. The encapsulated offer of the previous host is hashed together with the identify of the next host to form the current integrity check value. One may notice that each \( h_i \) links all the previous offers in the chain, while pointing to the next host. Therefore if an attacker replaces \( O_i \) by \( O_i' \) at an arbitrary point in the chain then the chain would be invalidated because: \( H(O_i, S_i) \neq H(O_i', S_{i+2}) \). This would be true even if the attacker would be the host \( S_i \) that by some methods has managed to store the signature the secret key \( y \), because: \( SIG_y(ENC_y(o_i', r_i), h_{i+1}, y_{i+1}) \neq SIG_y(ENC_y(o_i, r_i), h_i, y_{i+1}) \). Therefore, strong forward integrity also holds.

Insertion resilience is also satisfied because of the collision-free hash function used for computing the integrity check values. An insertion of \( O_i' \) at point \( i \) in the chain would invalidate the chaining relation after this point because: \( H(O_i', S_{i+1}) \neq H(O_{i-1}, S_{i+1}) \). Moreover, this insertion would force the attacker to use the same secret/public key pair generated by the previous host that posted an offer in the chain, i.e. \( S_{i-1} \), this being the only way to preserve the integrity of the chain of offers. Assuming that the attacker may somehow obtain these values, the originator may easily detect such a highly unlikely situation (i.e. two consecutive host generating the same secret/public key pair), and invalidate the chain of offers.

Truncation resilience holds at any point \( i \) in the chain except if the attacker has access to the secret key \( y_{i+1} \). In this case the attacker can perform a truncation at \( i \) and then add one or more offers.

## 5 Attacks

Karjoth et al. [6] explicitly state that forward integrity only holds if the attacker does not modify the last encapsulated offer from the current state of the chain \( (O_0, O_1, ..., O_m, m < n) \). An attacker is also able to truncate the chain at some point and grow a fake stem. Truncation is acknowledged as the most difficult problem that free-roaming mobile agents face. Roth [9] credits Karjoth et al. for clearly stating this vulnerability and describes how an attacker may exploit it.

### 5.1 Attack against P1

Roth [8] calls the attacker Eve, denoted by \( E \). She receives the agent (denoted II) along with its chain of encapsulated offers \( (O_0, O_1, ..., O_m, m < n) \), and picks any point \( j \) with \( 0 < j < m \). Suppose Eve is employed by one of the shops \( S_k \), \( k < j \). A truncation is performed at point \( j \). In order to maintain the chaining relation valid, Eve needs to collect an offer from the original host \( S_j \). However, she does not send the original agent (II) to collect the offer. Instead, Eve make of few changes to II obtaining a slightly different agent, denoted II_E. For instance consider that agent II had been programmed to gather offers for a certain laptop configuration in the price range of 500-
2000 EUR. Eve’s agent $\Pi_E$ is identical to $\Pi$ except it is set to search for laptops in the price range of 1700-2000 EUR. This will filter out a series of attractive offers that $\Pi$ would have collected. Since $\Pi_E$ is controlled by Eve, she fixes $\Pi_E$’s itinerary such that it takes the following trip:

$$E \rightarrow S_j : \{O_i | 0 \leq i \leq j - 1\}$$

$$S_j \rightarrow S_{j+1} : \{O_i | 0 \leq i \leq j\}$$

$$S_{j+1} \rightarrow E : \{O_i | 0 \leq i \leq j + 1\}$$

An important thing to note is that $S_{j+1}$ was chosen by Eve. After these three steps, $\Pi_E$ has collected some unattractive offers from shops $S_j$ and $S_{j+1}$. However, the chain is not valid because $h_{j+1}$ points to Eve. Therefore, she can throw away $O_{j+1}$, then increment $j$ and perform the three steps again, choosing from $S_k$’s biggest competitors, which she wants the agent ($\Pi_E$) to visit next. Eve can repeat these steps how many times she wishes, obtaining each time an extra set of offers forming a valid chain. After having gathered a sufficient amount of unattractive offers, she gives the chain of encapsulated offers to the original agent $\Pi$ and sends it on its normal path. The agent will normally not visit a shop from which it has already obtained an offer. When the agent returns to the originator, this attack will not be detected. Now it is more likely that the originator will pick one of $S_k$’s products, than in a normal run of the agent. Roth notes that the main problem in this attack is that the shops can be abused as oracles for generating offers to the terms set by Eve, rather than those set by the originator.

5.2 Attack against P2

An important observation made by Roth [9] is the fact that Karjoth et al. do not mention how a host can determine to originator’s identity when using protocol P2. The only indication of the originator’s identity arises from it’s public key that is used to encrypt the signed offer. However this does not authenticate the originator, since anyone could have used his publicly available key to encrypt data.

The attack is once again carried out by Eve upon receiving the agent $\Pi$ and its chain of encapsulated offers $(O_0, O_1, ..., O_m, m < n)$. Eve makes a truncation at any chosen point $j < m$. Afterwards she uses her own agent $\Pi_E$ to collect several offers in her name, from a set of shops she choses, but which has to include $S_{j+1}$. Upon receiving the offers she can decrypt them using her private key. Eve then begins generating random nonces for each offer (starting with the one from $S_{j+1}$) and encrypting them with the public key of the originator. At each step she has sufficient information to compute the integrity check values such that the chain of offers will be valid. The final integrity check value comprising the chaining relation will be computed using the identity of the shop to which Eve wants the agent to go to next.

After the agent finishes gathering offers, it will return back to the originator with a valid chain of encapsulated offers. The example scenario from the previous attack, with the same attacker goal, also applies here. Roth points out the fact that the problem is not due to the protocol’s vulnerability to truncation attacks, but due to Eve’s ability to set the terms for offers returned to the originator.

5.3 Attack against P3 & P4

The attack on protocol P4 described by Roth is a truncation attack. The attacker, Eve receives the agent $\Pi$ together with its chain of encapsulated offers $(O_0, O_1, ..., O_m, m < n)$. Eve is employed by $S_k$, $k < m$ who provides her with the secret key $(Y_{k+1})$ it has generated for the shop following it in the itinerary of the agent. Eve can perform a truncation at $k$ and then generate a series of fake offers. Since the public/secret key pairs are randomly generated by every shop, authentication of the shops is lost. This means that Eve can generate how many key pairs it wants to construct a desired number of fake offers. This can be done due to the fact that offers are forgeable using this signature scheme. At the end Eve adds the host to which she wants to send the agent to the final integrity check value and sends the chain of encapsulated offers on a public channel and the last generated private key on the confidential channel to the same host.

Upon the agent’s return to the originator this attack will not be detected. Note that in this case the attacker did not need to gather signed offers from other hosts. The offers were simply created on the machine of the attacker. Also note that the same attack may be executed on protocol P3, with the difference that the attacker needs the value $h_{k+1}$ from the malicious host $S_k$.

6 Solutions

The attacks described in the previous section, all exploit the same vulnerability, i.e. shops may be abused as oracles for signing offers to the terms set by the attacker and not the originator. This is very effective in the context of
a truncation attack and growing a fake stem. Free-roaming agents are particularly vulnerable to these types of attacks because they are intended to run autonomously, i.e. without human interaction.

Karjoth et al. [6] briefly mention three possibilities of how truncation attacks may be overcome. First, an agent could embed a partial precomputed itinerary in its encapsulated offer, which would be checked upon the agent’s return. However, this approach requires a way of handling shops which are offline. Otherwise an agent run that encounters an offline shop would be discarded by the originator. The second solution would be to broadcast an agent’s itinerary after the agent arrives back to the originator. However, this approach would not be appropriate for privacy preserving protocols. The third solution requires the function for computing the agent’s itinerary, be verifiable by the originator. Upon the agents return the originator could check if the itinerary respected this function. However, this approach is not extremely flexible, being comparable to a fixed itinerary. An agent with a fixed itinerary is not vulnerable to the attacks described in the previous section.

The proper use of cryptography in mobile agent protocols should allow an originator to verify if the offers collected by the agent were collected by the agent itself or added by an attacker. Roth [9] describes two approaches for solving the previous problem. These two solutions will be presented in the following subsections. Note that the two solutions can and should be used in combination. The first solution is meant to authenticate the true originator of the agent to a legitimate host. The second solution allows the originator to check whether an offer was intended for the corresponding agent he sent (Π), or some other agent of the attacker (Π).

6.1 Binding confidential data

The idea behind this approach is to program the agent such that it allows an honest host to detect whether the encapsulated chain of offers really belongs to the agent. In the following formal description, \( \{m\}_{K} \) represents a symmetric encryption of the message \( m \) with secret key \( K \). The protocol modification described by Roth is:

\[
S_{t} \rightarrow S_{t+1} : \quad SIG_{0}(Π, r_{0}, H(VERIF_{0}, K_{0})), \\
\{O_{j} | 0 \leq j \leq i\}_{K_{0}}, \\
ENC_{k+1}(K_{0})
\]

where \( K_{0} \) is a symmetric key generated by the originator, as well as the nonce \( r_{0} \). The originator’s identity can be derived from the associated, public verification key \( VERIF_{0} \), available in a PKI. The signature has the role of binding the agent source code together with a random nonce and a hash of both the verification key and the symmetric key.

Each host that is visited by the agent first extracts the secret key \( K_{0} \) from the third part of the message, which is encrypted with that host’s public key. The signature from the first part of the message is then verified and the identity of the originator is determined. The host then checks if the hash inside the signature is indeed equal to the hash of the key it extracted earlier and the originators public signing key. If they match then the host decrypts the ciphertext containing the chain of encapsulated offers and add its own offer. Afterwards, the agent encrypts the new chain with \( K_{0} \), then it encrypts \( K_{0} \) with the public key of the following host that will be visited.

This approach protects the agent from external attacks performed by capturing the agent on its way between two hosts. An external attacker cannot append offers to the chain since it does not know \( K_{0} \). However, if the originator’s agent willingly chooses to visit a malicious host, then the integrity of the chain is compromised. This happens because one the malicious host obtains the secret key \( K_{0} \), it can tailor its own agent to gather offers that it may append to the current chain of the honest agent and then encrypt then using \( K_{0} \).

6.2 Binding acquired data

The second solution described by Roth is more effective against the attacks described in Section 5. The idea behind this approach is that each hosts that makes an offer, signs that offer together with a unique identifier of the agent. This solution is only applicable if the signatures of the hosts cannot be forged by an attacker (i.e. in protocols P1 and P2). Roth also warns that digitally signing the source code of an agent is not enough to distinguish different runs of the same agent. Therefore he suggests that a random token \( o_{0} \) be signed along with the agent code. Furthermore, he derives the unique identifier of each agent run by hashing such a signature and calls the result an implicit name:

\[
\phi = H(SIG_{0}(Π, o_{0})).
\]

This approach requires the originator to include a signature of the agent’s code together with a random token \( SIG_{0}(Π, o_{0}) \) in the information that is sent in each step of the proto-
col. After a host verifies the signature, it makes an offer and then signs the offer together with the hash of $SIG_0(\Pi, o_i)$. Hence, the encapsulated offers for protocols P1 and P2 will have the following form: $SIG_i(ENC_0(o_i, r_i), h_i, \phi)$, respectively $ENC_0(SIG_i(o_i, \phi), r_i), h_i$, for $0 \leq i \leq n$. Therefore, an offer is only valid in the context of the agent instance whose implicit name is signed along with that offer. This acquired data binding can be easily verified by the originator.

This solution prevents the attacker from abusing shops as oracles for making offers based on some other terms than the originator’s. Changing the agents terms would require modifications in II and also signing the resulting agent with the originators private signing key. This is assumed to be extremely difficult to do, however even if the attacker succeeds in doing this, the implicit names of the two agents will be different because we have also assumed a collision-free hash function.

7 Conclusions

This paper has presented several security protocols designed for mobile agents. Yee’s work [10] was used as a reference for the four protocols developed later by Karjoth et al. [6] for free-roaming agents. The first two protocols described by Karjoth et al. strongly rely on public key signatures and corresponding certificates available in a PKI. This aspect enables non-repudiability and unforgeability of offers.

The last two protocols in [6] are based on secret hash chains and do not require a PKI. However, these two protocols require the exchange of hash values over a confidential channel between every pair of adjacent hosts in the agent itinerary, which may be expensive to establish.

Each protocol was formally described and analyzed with regard to the security properties given by Karjoth et al. A remarkable achievement of the four protocols proposed in [6] is that they all satisfy the strong forward integrity property, insertion resilience and data confidentiality, which were not all satisfied by the protocols proposed by Yee.

The forward integrity property has the constraint that the attacker does not change the last encapsulated offer from the chain. Lifting this restriction implies that an attacker has the possibility to perform truncation attacks and to grow a fake stem. This vulnerability is exploited by the attacks described by Roth [9, 8], who also points out the fact the hosts may be abused as oracles to generate signed offers according to the terms set by the attacker, which can be added to the original chain of encapsulated offers without invalidating it.

Roth also suggests that the authors of mobile agent protocols follow the advice given by Needham and Anderson [1] by which his paper was also inspired. The first principle he quotes is “be careful, especially when signing or decrypting data, not to let yourself be used as an oracle by the opponent”. This is exactly what happens in the attacks presented by Roth. However, his solution described in Section 6.1 of this paper addresses this issue by enabling the hosts to identify the originator of the agent. The second principle quoted by Roth is: “where the identity of a principle is essential to the meaning of a message, it should be mentioned explicitly in that message”. In the context of the protocols described in Sections 3 and 4, this principle is not respected because upon the agent’s return, the originator cannot distinguish offers gathered by its agent or another agent created by some attacker. This aspect is also treated by the second solution proposed by Roth, which was presented in Section 6.2. The solution implies that each host should sign a unique identifier of the agent instance along with the offer it makes. Agent instances are distinguished from each other by signing a different random nonce in each protocol run, along with the source code of the agent.

In addition to Roth’s suggestions we would also make a recommendation for protocol P1. Each host should encrypt its own identity together with the offer it makes. Signing this encryption afterwards together with the integrity check value would also ensure that the originator is certain which shop made which offer. Even though it is highly unlikely that a shop will blindly sign a ciphertext containing an offer made by someone else, this may prove to be useful in some other context. For instance if the originator has an important problem and wants to offer an attractive prize to anyone who can solve it. It sends the agent to gather answers and at the end it inspects the best solution. In this scenario an attacker might strip the signature of some legitimate host who she assumes has given the best answer, and ads her own signature on the ciphertext. In the original version of the P1 protocol, the originator will then think that this answer actually came from the attacker. However, using our suggestion the identity of the host that gave the answer would be safely encrypted with the originators public key. This attack would not be possible since if the attacker’s signature will not correspond to the identity encrypted with the answer.
References


