# A Beehive Approach to e-Commerce Mobile Agents

Martin van Hensbergen
Fox-IT
Forensic IT Experts
Delft
The Netherlands
hensbergen@fox-it.com

Bartek Gedrojc, Jan C.A. van der Lubbe
Delft University of Technology
Fac. EEMCS, ICT Group
Delft
The Netherlands
{B.Gedrojc, J.C.A.vanderlubbe}@tudelft.nl

**Abstract**

In this paper we introduce a practical mobile agent scenario which could be used as a basis for an e-commerce setting. Our approach is not based on a centralized trusted environment but on the use of multiple agents; analogue to a beehive where there is one queen who makes the decisions and multiple drones that do the work. Our results provide privacy of the itinerary of the mobile agents, protects the agents' code and the query. We also provide a mechanism to skip certain hosts without the intervention of the user.

## 1 Introduction

We consider the problem of conducting e-commerce in a secure way using mobile agents in an untrusted environment. Mobile agents are autonomous pieces of software that run on remote hosts in order to carry out a task on behalf of its user. The code of the agent itself, together with some other data, is transported over the internet as opposed to only data. The agents we consider operate in an e-commerce environment where the goal is to purchase a desired item for a user. They visit a collection of hosts, which can be malicious, curious or honest, that may or may not sell the article in question, ask them for an offer and eventually decide which offer is the best. This best offer is then digitally signed by the agents to commit themselves to the offer made by the host.

### 1.1 Security issues and related works

Since the agents are executed on other computers than that of the user, namely those of the merchants, the execution of the agent may be tampered with to bias the output of the agent in favor of the merchant. Therefore, the agent may want to hide certain information from the merchant, like its maximum price it is willing to pay for an item and which other merchants it will ask for a bid, in order to give as little information away as possible. Of the possible attacks that malicious hosts can mount, the internal replay attack is one of the most easy to realize attacks. Especially if the agent outputs some value (e.g. whether or not it agrees to an offer made) to the host. In this case, the malicious host can use the agent as an oracle to extract private information (e.g. the decision-logic and parameters) from it by executing it a numerous times with different inputs and observing the outputs. For a very concise review on the possible security risks with agents, please refer to [12, 13].

Mobile agent technology is blending of a number of technologies, in particular artificial intelligence and mobile code. Mobile code is the transfer of code or function call, from one host (sender) to another (the recipient), which is executed on the recipient's side. It is often used in cases where band-width limitations make it more efficient to send code to the data than vice versa. In combination with artificial intelligence, mobile code may be more autonomous and may travel a more complex path to execute more complex functions.

The transfer of code from one computer to another computer and the execution thereof poses a number of security issues. The two main categories of these issues are the malicious code problem and the malicious host problem.

In the first category we find problems that deal with protecting the executer of the mobile code from the sent code. As an example, the recent numerous outbreaks of computer worms and trojans [5] show that executing malicious software can have enormous negative consequences. A number of techniques have been suggested, and used, to alleviate some of these issues, like sandboxing [8] and the use of certificates which states what agents should be able to do or not do [16].

The problems in the latter category concern with the successful execution of the sent code by a host, which you may or may not trust. Since the mobile code is transported to hosts that you may not trust and may have incentives to manipulate or spy on the code or data the agent carries, protection mechanisms must be in place to prevent, or detect, tampering of code, replaying the code multiple times (to clone it or observe behavior) or spying out of certain secret data carried by the code.

Much literature is available which deals with different aspects of the malicious host problem [3, 1]. Although not many papers deal with privacy of itineraries of multi-hop agents, different approaches are known regarding the prevention of altering itineraries. Ensuring that an agent transports itself to the proper next destination can be done e.g. by Mutual Itinerary Recording [11], whereby multiple agents validate each-other's itinerary. On moving on to the next host in its itinerary, an agent relays the previous destination, current destination and next destination -through an authenticated channel -to another agent on another host, which records this and warns for any discrepancies. Drawbacks of this scheme include the cost of setting up such an authenticated channel, as well as the communication overhead through that channel.

Some suggest a notion like a Trusted Agent Proxy server [6], which is a trusted middleman server that anonymises authenticated agent entities in agent itineraries and providing a trusted base from which agents are sent to untrusted environments. Having a trusted element within your malicious environment is a good way of mitigating risks but it is quite a stringent requirement.

Another method of protecting the agent's code from tampering and inspection, as suggested in [10]and [9], is to use trusted hardware in the agent platforms which can execute encrypted functions. But the cost of requiring such hardware can be prohibitive.

The use of code-signing schemes, as used in e.g. Agent TCL [7], is also a recognized method to prevent alteration of the mobile agent's code. But unfortunately, only the code is authenticated by the author, while agents usually take different parameters that are specified by the user of the agent and not the author. These parameters should also, somehow, be authenticated and protected.

In this paper, a number of issues are addressed in the malicious host setting which deal with the protection of agent's itinerary, preventing replay attacks and minimize the risk of loss of private data.

# 2 The agent model

In this section we will define our problem statement in section 2.1. Then we give an high-level description of our approach in section 2.2 and in section 2.3 discuss the main cryptographic techniques which will be used in the solution to the problem.

## 2.1 Problem statement

Rather than concentrating on one particular problem of the agent's execution, a number of different aspects of the security of agents are addressed.

An agent consists of, among other things, the agents' code, an itinerary and a query. The code is the program itself which will be executed by the host. The itinerary is a list of hosts

that the agent will visit in order to obtain information. The query describes what the agent is looking for on behalf of the user.

Furthermore it is required that the itinerary, query and code of the agent be protected from malicious hosts. Malicious hosts should not be able to change any of these without being detected before signing the final bid.

We want to minimize the amount of network traffic that the agents generate, while at the same time let the agent visit as many of the hosts in its itinerary as possible. In order to minimize the traffic of the agent, it is assumed that the agent visiting the hosts is a multi-hop agent. For the travel route of the agents we do not allow a star topology, where the agent returns after each hop to a central host where it gets its new destination. The main reason being that, if the total number of hosts to visit by the agent equals $n$, this method requires $2n$ communication hops to complete its route. Rather, we want to let host $i$ in the itinerary send the agent to the next host $i + 1$ immediately so that the number of communication hops equals $n$. Especially for large $n$ this is beneficial.

In practical situations it is possible that a host $H_i$ in the agent's itinerary may not accept agents due to malfunctioning, being too busy, etc. Measures should be in place to deal with these situations such that if the agent is on host $H_{i-1}$ and cannot be sent to $H_i$, it can skip the malfunctioning host without human-intervention. However, this requirement should not imply that host $H_{i-1}$ should always know the locations beyond host $H_i$ or before $H_{i-2}$. A host should only know where the agent comes from and where it has to go next (because it has to send it to that host).

The internal replay attack being the most easy to mount attack, the agent model should prevent these attacks from being fruitful. Again, the actual replaying cannot be prevented but it should not be worthwhile to do so.

## 2.2 Approach

The key to our approach is the use of multiple agents for one goal with strict separation of tasks for each agent. Considering the fact that a decision process of selecting the best offer can only begin after all necessary information is collected, it seems natural to split the task of data collecting and decision making into two separate agents. Analogous to a beehive where there is one queen who makes the decisions and multiple drones that do the actual work, we shall speak of drone agents and queen agents.

These two types of agents can shortly be characterized as:

- **Drone** An agent that can only collect data but does not have any decision making logic
- **Queen** An agent that takes the output of the drone(s) and makes a decision based on that data

The greatest adversaries an agent faces are the malicious hosts, as these can mount any type of attack possible on the agent, whereas a curious host will execute the agent correctly but will try to learn any secrets it may finds. For this reason one doesn't want to expose an agent with decision logic to these malicious hosts as this would make the agent's secrets very vulnerable to attack. The gathering of information is a rather neutral activity however and sending only a drone to collect the information will keep the decision logic well away from the malicious hosts.

The queen, which does carry the decision logic, will not pass any malicious hosts. Rather, it is executed on a fixed - at most curious - host $HQ$ and is quite immobile compared to the drone. The queen will thus travel only to a curious host where it waits for the drone's arrival. Once the drone and queen are together they will make a decision on the best offer.

There is also a third agent involved, called the helper-agent, which will be involved when the drone cannot be sent to its next destination. This helper-agent will also reside on a curious host (which can either be the same or a different host as where the queen resides) but will not

move from there. The helper-agent is only needed when a host Hi in the drone's itinerary does not function, otherwise the helper-agent will just shut down after a pre-determined time.

We can thus define three layers of trust in our agent model. The first layer is the trusted layer; all hosts in this layer are completely trusted. Only the host of the user (the instantiator of the agents) is situated in this layer, which allows for the instantiation of the agent(s) to occur in a safe environment.

The second layer is a curious layer where the helper-agent and queen reside, while the third layer is the malicious layer where the drones operate. Agents do not move from the curious to the malicious environment and only the drones move from the malicious to the curious environment at the end of its data collection task. We assume that none of the hosts in the malicious environment collude with the hosts in the curious environment (although we will show in some cases what happens if this assumption is not valid).

Note that the above implies that one knows a-priori of the hosts in the curious layer that they are, indeed, curious and not malicious; something you don't always know in advance. The rationalization of the existence of such hosts can be by considering them as being an Agent Service Provider (ASP), much like there are Internet Service Providers. An ASP is an independent service provider that, maybe in exchange for a fee, hosts agents for subscribers. This ASP does not take part in the bidding scheme -it just provides computing power for agents to execute. As it is a service provider, it is not in its business advantage to tamper with its customers' agents so it is reasonable to assume that the agents are executed correctly. However, the ASP might be curious ( perhaps not as an entity, but maybe individual system administrators working for the ASP might not be trusted).

## 2.3 Cryptographic techniques

In order to protect different aspects of the agent model a variety of cryptographic techniques and concepts is needed. None of the protocols are described at algorithm level, which means that a suitable algorithm can be chosen whenever a hash-function or encryption algorithm is used. The encryption of a value $v$ with a symmetric encryption algorithm using key $K$ is denoted as $E_K(v)$, whereas the encryption of the value using the public key of a particular host $H_i$ is denoted as $E_{H_i}(v)$ (or $E_{P_{H_i}}(v)$). A digital signature over a value $v$ with host $H_i'$s private key will be denoted by $Sig_{H_i}(v)$.

**Secret Sharing Scheme.** For the storage of the secret parameters in the queen a public key encryption algorithm is used. Since this storage will need to be decrypted once the drone has reached the queen, a secret sharing algorithm is used to split the decryption key in two parts; one which is given with the drone and one which is given to the queen e.g. [14, 2]. Without the presence of the drone, the host which executes the queen cannot decrypt the secret data stored in the queen.

**Threshold Signature Scheme.** Since the agents need to sign the best bid at the end of the journey, a signing key must be stored by the agents. For obvious reasons this key cannot be placed in the drone. Placing the signing key in the encrypted storage of the queen, while better suited, has the drawback that the signing key will be reconstructed in its entirety on a curious host.

To counter this a threshold signature scheme is chosen e.g [4]. With a threshold signature scheme, the signing key is split into two parts. Each party in the signature process can use its share to create a partial signature on a message. These partial signatures can then later be combined to create one complete signature. In our model we let the merchants partially sign their bids all with the same share and give that to the drone. Once the queen finds the best offer, it creates a partial signature of that same offer with its share and combines the two partial signature to one complete signature. This way, the complete signing key is never reconstructed in one place.

**Hash Chaining.** Is a method of providing integrity of data when this data is being augmented by multiple parties. The within this section described hash chaining technique is taken from [15]. The protocol described here will need some modifications for our agent model, but the principles will remain the same. Each host $H_i$ that the drone visits will add its offer $o_i$ to an encrypted storage in the agent. By using hash chaining, host $H_i$ not only adds its offer to the storage, but also makes a commitment that he adds it to the proper storage, by including a hash of the previous storage state. Each host $H_i$ will follow the following protocol for storing its offer $o_i$ to the storage.

- Encapsulated offer:
$$O_i = Sig_{H_i}(E_{P_Q}(o_i, r_i), h_i), 0 \leq i \leq n \quad (1)$$
- Chaining relation:
$$h_0 = h(o_0, H_1) \quad (2)$$
$$h_i = h(O_{i-1}, H_{i+1}), 1 \leq i \leq n \quad (3)$$

Here, $o_0$ is initial information (e.g. identitiy of the agent), $o_i$ the offer of host $H_i$, $O_i$ the encapsulated offer from host $H_i$, $r_i$ a random number generatated by $H_i$, $E_{P_Q}(v)$ is the encryption of value $v$ with the public key of the queen and $h(.)$ a cryptographic hash function. The encrypted storage will consist of the chain $O_0, O_1, \ldots, O_n$.

The essence of the protocol is that a host $H_i$ signs both its offer and a hash value taken over the last encapsulated offer and the next destination of the agent. If a malicious host $H_i$ would like to delete, for example, an offer $O_k(k < m)$, from the storage, then this will be detected during verification of the hash chain because the committed value $h_{k+1}$ will not verify.

# 3 Protocol

This section will describe the protocols to instantiate the drone, queen and helper agents, the bidding protocol, the help-protocol and the decision protocol. The complete process of creating agents, sending them out and gaining the results is called a mission.

## 3.1 Instantiation of the Agents

For each mission, three agents will be constructed. Let the itinerary of the drone be given by the hosts $H_1, H_2, H_3, \ldots, H_n, HQ$ where $H_i(1 \leq i \leq n)$ are merchants and $HQ$ is the location of the host which executes the queen. Let $H_P$ denote the location of the host which hosts the helper-agent. Fix a security parameter $m \leq 1$ which denotes the maximum amount of succeeding hosts that may fail so that the drone can still continue its journey. The case $m = 0$ does not require a helper agent so we ignore this case. Denote the parameters which describe the item(s) that the agent seeks with $Q$ and the secret decision parameters or logic by $F$.

**Mission instantiation.**

- Generate a public key $P_Q$, which will be used to encrypt the Queen's data, and split the corresponding private key in two parts $S_{Q_1}$ and $S_{Q_2}$ using a secret sharing scheme.
- Generate a secret signing key $S$, which will be used to sign the best offer by the merchants, and split the signing key in two parts $s_1$ and $s_2$ using a threshold signature scheme.

**Drone instantiation.** In this protocol, $h$ is a strong cryptographic hash function which generates $d$ bit hashes (with $d$ sufficiently large) and $r|_k$ means the first $k$ bits of the bit-string $r$.

1. Choose a $k$ such that $2^k = |\{H_i\}|$ and $k < d/2$.
2. For each host $H_i$ to visit, generate a symmetric encryption key $K_i$ and a (small) random nonce $r_i$ $(1 \leq i \leq n)$, such that $r_i|_k \neq r_j|_k$ for $j < i$ and a random nonce $n_i$.
3. Calculate iteratively the itinerary as

$$I_n = HQ \tag{4}$$
$$I_i = [E_{P_{H_i}}(K_i, r_i), E_{K_i}(H_{i+1}, S_{Q_2}, I_{i+1})] \text{ with } i = n-1, \ldots, n-m \tag{5}$$
$$I_i = [E_{P_{H_i}}(K_i, r_i), E_{K_i}(H_{i+1}, I_{i+1})] \text{ with } i = n-m-1, \ldots, 1 \tag{6}$$

Store $I_1$, $P_Q$, $S_{Q_2}$ and the location of the helper-agent in the drone and send it to $H_1$.

**Queen instantiation protocol.**

- Calculate the encrypted storage: $E_{P_Q}(H_1, n_1, H_2, n_2, \ldots, H_n, n_n, F, s_2)$
- Store the encrypted storage together with $S_{Q_1}$ in the queen

**Helper agent instantiation protocol.** Calculate and store the following lookup-table:

Table 1: Helper agent lookup-table.

| Look-up key | value | verification |
|---|---|---|
| $r_i\|_k$ | $E_{K_i}(E_{P_{H_i}}(K_{i+t}, n_{i+t}))$ | $h_{i,t} = h(r_i \|\| H_{i+t})$ |

with $t = 1, \ldots, m$ and $i = 1, \ldots, n-1$.

## 3.2 Execution of the Agents

When the agents are initialized, the queen and helper-agent are sent to their respective agent service providers. The helper-agent awaits instantiations of the helper protocol and is deactivated after a pre-determined period, so helper agents will not leave the ASP once they are there. The queen awaits the coming of the drone or gets deactivated after a pre-determined period whichever comes first. The drone is sent to $H_1$ and continues its journey from there.

**Offer collection protocol - drone at merchant.** When a drone is executed on a host $H_i$, the following protocol is executed

1. Host $H_i$ uses its private key to decrypt the first element of $I_i$, as given in (**??**) and (**??**), to obtain $K_i$, which can be used to decrypt the second element of $I_i$ to obtain $H_{i+1}$ and the rest of the (encrypted) itinerary.
2. Host $H_i$ creates an offer $o_i$ and uses $s_2$ to partially sign its offer, denoted by $c_{i,2}$
3. Host $H_i$ calculates using $PQ$ the encapsulated offer using the following hash chaining relations:

   If help-protool was not needed:

$$O_i = Sig_{H_i}(E_{P_Q}(o_i, \hat{r}_i, c_{i,2}), h_i), 0 \leq i \leq n \tag{7}$$
$$h_i = h(O_{i-1}, H_i) \tag{8}$$

If help-protocol was needed to skip $t$ hosts:

$$O_i = Sig_{H_i}(E_{P_Q}(o_i, \hat{r}_i, c_{i,2}), n_{i+1}, n_{i+2}, \ldots, n_{i+t}), h_i) \quad (9)$$

$$h_i = h(O_{i-1}, H_{i+t+1}) \quad (10)$$

4. The hash chain, $I_{i+1}$ and agent are sent to the next destination.

In case step 4 fails, the help of the helper-agent is called and the following help-protocol is used:

**Help protocol - drone at merchant communicating with helper-agent.**

1. $H_i$ sends a help request to the helper agent consisting of the tuple $r_i, \hat{H}_1, \hat{H}_2, \ldots, \hat{H}_s$ where $r_i$ is the value included in $I_i$ and $\hat{H}_1, \hat{H}_2, \ldots, \hat{H}_s$ are the hosts it wishes to skip
2. $P$ checks if $1 \leq s \leq m$ and if $r_i|_k$ is part of its lookup table. If not, it aborts.
3. For each $k = 1, \ldots, s$ execute step 4
4. $P$ verifies if $h(r_i||\hat{H}_k) = h_{i,k}$
5. $P$ returns $E_{K_i}(E_{P_{H_i}}(K_{i+s}, n_{i+s}))$ to $H_i$

**Decision protocol - drone and queen at $HQ$.** Once the drone reaches the queen, they will together decide on the best offer. For this to work, the following steps must be undertaken:

1. $S_{Q_1}$ from the drone and $S_{Q_2}$ from the queen are combined to obtain $S_Q$
2. The queens encrypted storage is decrypted using $S_Q$, thereby obtaining the decision logic, the other half of the signing key $s_1$ and the itinerary that the drone should have followed
3. For each encapsulated offer that the drone has collected execute steps 4 - 6
4. Decrypt $O_i$ to get actual offer $o_i$
5. Verify if signed by a host which was included in itinerary, if verification fails: abort
6. Verify if each signer is only represented once
7. For each host $H_i \in I_D$ that did not do an offer, verify if $n_i$ is present in encrypted storage. If verification fails: abort.
8. Input the offers $o_1, o_2, \ldots, o_n$ into $F$ to determine $b$ offer $o_b$
9. Use $s_2$ to partially sign $o_b$ and combine with $c_{b,1}$ to make the signature complete.

# 4 Conclusion

In this paper we have introduced a practical mobile agent scenario which could be used as a basis for an e-commerce setting. By defining strict tasks for each agent, we prevent sensitive data from being exposed to the possibly malicious merchants during the bidding process. Even though we use a multi-hop agent with fixed itinerary to travel to the different merchants, merchants see only a very small part of the agent's itinerary. In case of failures in the agent's itinerary, a help protocol can be executed to overcome this difficulty. If all of the hosts function properly this help protocol is not needed. Protocols are described in a general so that different algorithms could be used depending on other (performance and storage) factors.

## References


[1] J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. In *Proceedings of the IEEE Symposium on Security and Privacy*, page 2. IEEE Computer Society, 2001.

[2] G.R. Blakely. Safeguarding cryptographic keys. In *Proceedings AFIPS 1979 National Computer Conference*, pages 313–317. AFIPS, 1979.

[3] C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-round secure computation and secure autonomous mobile agents. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 512–523. Springer-Verlag, 2000.

[4] Y.G. Desmedt and Y. Frankel. Perfect homomorphic zero-knowledge threshold schemes over any finite abelian group. *SIAM J. Discret. Math.*, 7(4):667–679, 1994.

[5] G. Gebhart. Worm propagation and countermeasures. The SANS Institute Security Reading Room, 2004.

[6] M. Giansiracusa. Mobile agent protection mechanisms, and the trusted agent proxy server (taps) architecture. Technical report, Information Security Institute (ISI), 2003.

[7] R.S. Gray. Agent tcl: A flexible and secure mobile-agent system. Technical report, Hanover, NH, USA, 1998.

[8] C. Lai, L. Gong, L. Koved, A. Nadalin, and R. Schemers. User authentication and authorization in the Java platform. In *15th Annual Computer Security Applications Conference*, pages 285–290. IEEE Computer Society Press, 1999.

[9] H. Lee, J. Alves-Foss, and S. Harrison. The construction of secure mobile agents via evaluating encrypted functions. *Web Intelli. and Agent Sys.*, 2(1):1–19, 2004.

[10] S. Loureiro, L. Bussard, and Y. Roudier. Extending tamper-proof hardware security to untrusted execution environments. In *CARDIS'02, 5th IFIP/USENIX International Conference on Smart Card Research and Advanced Applications, November 21-22, 2002 - San Jose, USA*, Nov 2002.

[11] Volker Roth. Mutual protection of co-operating agents. In *Secure Internet Programming*, pages 275–285, 1999.

[12] T. Sander and C. F. Tschudin. Protecting mobile agents against malicious hosts. *Lecture Notes in Computer Science*, 1419:44–60, 1998.

[13] T. Sander and C. F. Tschudin. Towards mobile cryptography. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 1998. IEEE Computer Society Press.

[14] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[15] D. Singelée and B. Preneel. Secure e-commerce using mobile agents on untrusted hosts. Technical report, COSIC Internal Report, 2004.

[16] H. K. Tan and L. Moreau. Certificates for mobile code security. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 76–81, New York, NY, USA, 2002. ACM Press.