On the Robustness of some Cryptographic Protocols for Mobile Agent Protection

Volker Roth

Fraunhofer Institut für Graphische Datenverarbeitung
Rundeturmstraße 6, 64283 Darmstadt, Germany
vroth@igd.fhg.de

Abstract. Mobile agent security is still a young discipline and most naturally, the focus up to the time of writing was on inventing new cryptographic protocols for securing various aspects of mobile agents. However, past experience shows that protocols can be flawed, and flaws in protocols can remain unnoticed for a long period of time. The game of breaking and fixing protocols is a necessary evolutionary process that leads to a better understanding of the underlying problems and ultimately to more robust and secure systems. Although, to the best of our knowledge, little work has been published on breaking protocols for mobile agents, it is inconceivable that the multitude of protocols proposed so far are all flawless. As it turns out, the opposite is true. We identify flaws in protocols proposed by Corradi *et al.*, Karjoth *et al.*, and Karnik *et al.*, including protocols based on secure co-processors.

Keywords: mobile agent security, cryptanalysis, breaking security protocols.

1 Introduction

Analyzing cryptographic protocols for mobile agent protection means meeting old friends and foes. In [1, 2], Abadi, Needham, and Anderson summarized some rules and principles of good and bad practice for designing cryptographic protocols. We show in this paper that their advice was not followed thoroughly in the design of some cryptographic protocols meant to protect mobile agents against certain attacks by malicious hosts. We first summarize the typical objectives of the protocols we analyze:

Objective 1 (Confidentiality) *Mobile agents shall reveal cleartext only while being on trusted hosts.*

Objective 2 (Integrity) *The agents shall be protected such that they can acquire new data on each host they visit, but any tampering with pre-existing data must be detected by the agent's owner (and possibly by other hosts on the agent's itinerary).*

The general objective here is to protect certain features of a mobile agent against malicious hosts. By assumption, the host of the agent's owner is always trusted. Some of the protocols address both objectives simultaneously, others address just one. All protocols are targeted at protecting *free-roaming* mobile agents. In other words, mobile agents that are free to choose their respective next hop dynamically based on data they acquired in the course of their execution.

Unfortunately, these protocols expose hosts in a way that allows an attacker to abuse them as oracles for generating protocol data. This enables attacks on cryptographic protocols devised in [3–6]. In some cases this leads to a complete compromise of the protocol's security objectives. In other cases the adversary is able to forge and replace subsets of the protocol data in a way that makes it impossible for an agent's owner to detect the tampering. The important observation here is not that protocol data acquired by agents can be truncated (some authors already acknowledge this possibility) but that the attacker can exercise control over the data returned by an agent.

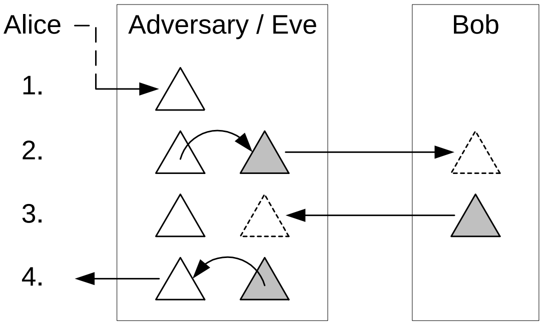

Fig. 1. Basic scheme of attacks we mount against various protocols. Triangles denote agents. Triangles shaded in gray denote agents created by the adversary Eve.

The attacks we mount on the analyzed protocols can best be described as *interleaving attack* [7, §10.5], which is "an impersonation or other deception involving selective combination of information from one or more previous or simultaneously ongoing protocol executions *(parallel sessions)*, including possible origination of one or more protocol executions by an adversary itself. Figure 1 illustrates the general scheme of attack: the adversary receives an agent, and copies protocol data back and forth between this agent and agents she sent herself.

2 Some Protocol Failures

We will write encryption of some *plaintext* into a *ciphertext* symbolically as $c = \{m\}_K$, where K is the *key* being used. A digital signature will be written as an encryption with a private signing key S^{-1}. We will write $S^{-1}(m)$ when we refer to the bare signature rather than the union of the signature and the signed data. We assume that the identity of the signer can be extracted from her signature. A cryptographic hash of some input will be written $h(m)$. Unless noted otherwise, we assume that h is *preimage resistant* and *collision resistant* [7, §9.2.2], which implies that h must also be *2nd-preimage resistant* [7, §9.2.5]. When A sends some message m to B we will write $A \rightarrow B : m$. We will write $A \rightarrow B : \{m\}_{K_{A,B}}$ when m is sent over a confidential channel. Concatenation of m_1 and m_2 is written as $m_1 \mid\mid m_2$. For ease of reading, we refer to some entities by their nicknames, e.g., Alice, Bob, and Eve. In general, Eve will play the role of the adversary, Alice will play the role of the victim agent's owner, Bob and Dave will play the role of additional entities taking part in the protocols. The itinerary of Alice's agent is written as $i_0, \ldots, i_n$, where $i_0 =$ Alice and i_n is the host currently visited by the agent.

2.1 Decrypting the targeted state

In [3], Karnik and Tripathi propose a *targeted state* as a means to protect the confidentiality of data carried by an agent. The idea is to make this data available to the agent only when it is on a host that is trusted with respect to keeping this data confidential from other agents and hosts. In order to achieve this, the plaintext is encrypted with the public key of the trusted host. The targeted state looks like this:

$$\{\{m_1\}_{K_{i_1}},\ \ldots,\ \{m_n\}_{K_{i_n}}\}_{S_A^{-1}}$$

The targeted state is signed by Alice, who is the originator of the agent owning the targeted state. Having received an agent, each host inspects the targeted state for ciphertexts it can decrypt. If so, the host decrypts it using its own private decryption key, and makes the cleartext available to the agent.

Below, we illustrate the attack on this protocol. Without loss of generality, we assume that the agent's targeted state contains a single ciphertext, which is encrypted with the public key of Bob. Alice first sends the agent to Eve from whom it hops to Bob and then returns to Alice. The protocol starts as follows (for simplicity, we assume here that an agent initially consists only of its targeted state and its program $\mathit{\Pi}_A$):

$$A \rightarrow E : \mathit{\Pi}_A,\ \{\{m\}_{K_B}\}_{S_A^{-1}}$$

The attack is straightforward. Eve strips off Alice's signature, copies $\{m\}_{K_B}$ into the targeted state of an agent of her own, signs this targeted state, and sends her agent to Bob:

$$\begin{aligned} E \rightarrow B : {}& \mathit{\Pi}_E,\ \{\{m\}_{K_B}\}_{S_E^{-1}} \\ B : {}& \mathit{\Pi}_E,\ \{\{m\}_{K_B}\}_{S_E^{-1}},\ \{\{m\}_{K_B}\}_{K_B^{-1}} = m \end{aligned}$$

Bob innocently decrypts the targeted state using his own private key and makes the resulting plaintext available to the agent. The agent then migrates back to Eve carrying the plaintext.

$$B \rightarrow E : \mathit{\Pi}_E, \{\{m\}_{K_B}\}_{S_E^{-1}},\ m$$

Eve now is in possession of the plaintext which should be available only to Bob; Alice never detects the attack. The problem with this protocol is that, due to a lack of redundancy in the ciphertext, Bob can be abused as an oracle. Alice needs to include an unforgeable identifier of her agent in the ciphertext, e.g., $h(\mathit{\Pi}_A, A)$ (see [8] for an alternative approach). Even then, the agent's program must be unique for each agent[1] and designed carefully such that it can not be abused in the way illustrated above by means of malicious state changes.

2.2 Forging the Append Only Container

In addition to the *targeted state*, Karnik and Tripathi also propose an *append only container*. The idea is to protect a container of objects in an agent such that new objects can be added to it but any subsequent modification of an object contained therein can be detected by the agent's owner. The protocol relies on an encrypted checksum, whose initial value $C_0 = \{r\}_{K_A}$ is computed by Alice (the agent's owner) based on a random nonce r. The nonce must be kept secret by Alice, and is used in the verification protocol upon the agent's return. The append only container is defined as follows:

$$\{\{m_1\}_{S_{i_1}^{-1}},\ \ldots,\ \{m_n\}_{S_{in}^{-1}}, C_n\}$$

Whenever a new object is appended to the append only container, the checksum is updated[2] as given below:

$$C_{n+1} = \{C_n \mid\mid S_{i_{n+1}}^{-1}(m_{n+1})\}_{K_A}$$

[1] Otherwise Eve can still cut & paste targeted states back and forth between agents that are owned by Alice and which share the same program.

[2] In the original protocol description, the signature and identity of the server are appended. On the other hand, we assume that the signer's identity can be extracted from the signature and appending it is, therefore, redundant.

The signer of the appended object is the host on which the append operation takes place. Upon the agent's return, Alice successively decrypts the checksums, extracts the signature, and verifies the signature against the corresponding object in the container. The last checksum must equal the initial nonce.

We now assume that Eve received Alice's agent and she knows C_j for some $1 \leq j \leq n$. Eve always knows C_n, because it is embedded in the container. She might collude with other servers which the agent visited before, or she might be part of a loop in the agent's itinerary. In these cases, Eve might discover a checksum C_j with $j < n$.

At this stage, Eve has multiple choices. She can either truncate the container up to the j'th object and grow a fake stem by releasing the agent. Or she can remove, add or replace arbitrary objects m_l with $l > j$ in the name of other hosts. In order to do this, Eve creates an agent with the object that she wants to add at $j+1$, and an append only container of her own, with checksum C_j as its initial value. Eve now sends her agent to Bob. There, Eve's agent inserts m_{j+1} in its own targeted state and migrates back:

$$\begin{aligned}
&E \rightarrow B : \mathit{\Pi}_E,\ m_{j+1},\ \{C_j\} \\
&B \rightarrow E : \mathit{\Pi}_E,\ \{\{m_{j+1}\}_{S_B^{-1}}, \{C_j \mid\mid S_B^{-1}(m_{j+1})\}_{K_E}\}
\end{aligned}$$

Upon the agent's return, Eve decrypts the checksum using her own private key, and re-encrypts it using the public key of Alice:

$$\begin{aligned}
C_{j+1} &= \{\{\{C_j \mid\mid S_B^{-1}(m_{j+1})\}_{K_E}\}_{K_E^{-1}}\}_{K_A} \\
&= \{C_j \mid\mid S_B^{-1}(m_{j+1})\}_{K_A}
\end{aligned}$$

Then, she constructs a new container:

$$\{\ \underbrace{\{m_1\}_{S_{i_1}^{-1}},\ \ldots,\ \{m_j\}_{S_{i_j}^{-1}}}_{\text{from A's agent}},\ \overbrace{\{m_{j+1}\}_{S_B^{-1}}, C_{j+1}}^{\text{from E's agent}}\ \}$$

which replaces the previous container of Alice's agent. This process is repeated with the new checksum until Eve is pleased with the result, and releases Alice's agent. Bob is not able to detect the attack, because C_j is not publicly verifiable (it is encrypted with Alice's public key). All Bob can see is the length of C_j, from which he can estimate the number of objects that must be in the append only container. So if Eve wants to make sure that Bob has no reason to get suspicious then she adds j signed objects to her agent's container before she sends it to Bob. As long as these objects are properly signed it does not matter who signed them and where she got them.

Once again, a lack of redundancy allows Eve to abuse other hosts as oracles, this time for the purpose of signing and checksum computation rather than decryption.

2.3 Forging the Multi-Hops Protocol

In [4], Corradi, Montanari, and Stefanelli propose a protocol they call *multi-hops*, which has the same purpose as the *append only container* presented by Karnik and Tripathi. It falls prey to the same type of attack. However, this time, the faked agent needs to do one more hop to complete its attack. For reference, we summarize the multi-hops protocol below.

Let $(\mathit{\Pi}, \mathcal{M}, \mathcal{P})$ be an agent where $\mathit{\Pi}$ is static (immutable) code and initialization data, $\mathcal{M}$ is (mutable) application data, and $\mathcal{P}$ is protocol data (meta information required by the protocol). Alice initializes her agent with $(\mathit{\Pi}_A, \epsilon, \epsilon)$. The protocol additionally requires a nonce γ and a message authentication code μ. The initial values are $\gamma_0 = h(r)$ and $\mu_0 = \epsilon$, where r is chosen randomly. On each hop, the agent can add some data m to its application data, which is then protected by the host using the multi-hops protocol. The protocol is defined as given below:

$$\begin{aligned} \gamma_n &= h(\gamma_{n-1}) \\ \mu_n &= h(m_n, \gamma_{n-1}, \mu_{n-1}, i_{n+1}) \\ \mathcal{P}_n &= \mathcal{P}_{n-1} \mid\mid S_{i_n}^{-1}(\mu_n) \\ \mathcal{M}_n &= \mathcal{M}_{n-1} \mid\mid m_n \mid\mid i_n \end{aligned}$$

$$i_n \rightarrow i_{n+1} \ : \ (\mathit{\Pi}_A, \mathcal{M}_n, \mathcal{P}_n), \{\gamma_n\}_{K_{i_{n+1}}}, \mu_n$$

The message authentication code μ_n serves as a *chaining relation* that binds results previously obtained by the agent to the ones obtained at the current host and to the identity of the agent's next hop.

Due to this chaining relation, the attack cannot be executed in the same way as it is done for the append only container. The resulting star shaped itinerary with Eve in the center would be too obvious in the protocol data. What Eve has to do here is to plan ahead one step.

Again, we assume that Eve is i_n, and she knows some γ_{j-1} for $1 \leq j \leq n$. She received the agent so she always knows γ_{n-1} and μ_{n-1}. She can obtain γ_{j-1} with $j < n$ by colluding with other hosts or as a result of loops in the agent's itinerary. Due to the chaining relation — remember that μ_{j-1} is computed on i_j — Eve does not have free choice of her first target, although she does have free choice for subsequent targets. In particular, if $j = n$ then she has to append an offer herself. Eve now chooses i_{j+1} and does the following:

$$\begin{aligned}
E \rightarrow i_j &: (\mathit{\Pi}_E, \mathcal{M}_{j-1}, \mathcal{P}_{j-1}), \{\gamma_{j-1}\}_{K_{i_j}}, \mu_{j-1} \\
i_j \rightarrow i_{j+1} &: (\mathit{\Pi}_E,\, \mathcal{M}_{j-1} \,||\, m_j \,||\, i_j,\, \mathcal{P}_{j-1} \,||\, S^{-1}_{i_j}(\mu_j)), \\
&\quad\; \{\gamma_j\}_{K_{i_{j+1}}}, \mu_j \\
i_{j+1} \rightarrow E &: (\mathit{\Pi}_E,\, \mathcal{M}_{j-1} \,||\, m_j \,||\, i_j \,||\, m^* \,||\, i_{j+1}, \\
&\quad\; \mathcal{P}_{j-1} \,||\, S^{-1}_{i_j}(\mu_j) \,||\, S^{-1}_{i_{j+1}}(\mu_{j+1})), \\
&\quad\; \{\gamma_{j+1}\}_{K_E}, \mu_{j+1}
\end{aligned}$$

Eve sends her agent first to i_j where it inserts some m_j chosen by her. Then it hops to i_{j+1} (chosen by Eve), inserts some random data m^* (which is discarded later on), and returns to Eve. Eve now updates Alice's agent as shown below, using the data acquired by her own agent:

$$(\underbrace{\mathit{\Pi}_A,\, \mathcal{M}_{j-1}}_{\text{A}} \,||\, \overbrace{m_j \,||\, i_j}^{\text{E}},\, \underbrace{\mathcal{P}_{j-1}}_{\text{A}} \,||\, \overbrace{S^{-1}_{i_j}(\mu_j)}^{\text{E}}) = (\mathit{\Pi}_A, \mathcal{M}_j, \mathcal{P}_j)$$

This completes the round. Eve now plans her next move (Eve chooses i_{j+2}, she already fixed i_{j+1} in the previous round). In order to send the agent to i_{j+1} she needs to know γ_j and μ_j, but she doesn't – yet. However, Eve knows γ_{j-1}, μ_{j-1}, and m_j. This is sufficient to compute

$$\begin{aligned}
\gamma_j &= h(\gamma_{j-1}) \\
\mu_j &= h(m_j, \gamma_{j-1}, \mu_{j-1}, i_{j+1})
\end{aligned}$$

At this stage, Eve either continues the attack, or she releases Alice's agent and sends it to i_{j+1}, where it resumes normal execution.

$$E \rightarrow i_{j+1} : (\mathit{\Pi}_A, \mathcal{M}_j, \mathcal{P}_j), \{\gamma_j\}_{K_{i_{j+1}}}, \mu_j$$

The underlying weakness of the multi hops protocol is the same as in the previously described protocols, namely, the abuse of servers as oracles. The digital signature gives no assurance about the intended recipient of the signed data.

3 The KAG Family of Protocols

Karjoth, Asokan, and Gülcü [5] published a family of protocols which are directed at preserving the integrity and confidentiality of data acquired by free-roaming agents. The general scenario is that of a comparison shopping agent that visits a number of shops, and collects offers from them. The idea behind these protocols is to preserve the integrity of collected offers. Some protocols also address confidentiality of offers.

3.1 Publicly Verifiable Chained Digital Signatures

The *Publicly verifiable chained digital signature protocol* (P1) is defined as given below:

$$\begin{aligned}
\mathcal{M}_n &= \{\{m_n, r_n\}_{K_A}, C_n\}_{S_{i_n}^{-1}} \\
C_n &= h(\mathcal{M}_{n-1}, i_{n+1}) \\
\mathcal{M}_0 &= \{\{m_0, r_0\}_{K_A}, C_0\}_{S_A^{-1}} \\
C_0 &= h(r_0, i_1)
\end{aligned}$$

$$i_n \rightarrow i_{n+1} \ : \ \mathit{\Pi},\ \{\mathcal{M}_0,\ \ldots,\ \mathcal{M}_n\}$$

where m_0 is a dummy offer, r_n is random salt that makes it harder to attack the encryption. C_n is called the *chaining relation* at n. By assumption, it shall be possible to deduce the identity of the signer from a signature [5, pp. 198]. The signer of $\mathcal{M}_0$ is deemed to be the owner of the agent (unfortunately, the authors of [5] do not explicitly mention from what they conclude who the owner of a given agent is, so we have to do a little guesswork here).

The security of the protocol relies on the assumption that an attacker does not change the last element $\mathcal{M}_n$ in the chain. However, there is no reason why an attacker would be so obliging. On the contrary, if the attacker is willing to build a complete chain for the agent then he can even remove chain elements *before* his own entry (this contrasts with e.g., the *honest prefix* property introduced by Yee [9, pp. 267]). The important observation here is that the input to all previous chaining relations is known.

We assume that Eve received an agent owned by Alice. Let Eve be $i_n, n > 1$. She picks j with $0 < j < n$ and a new i_{j+1} of her choice. Please note that there is no free choice of i_j once j is fixed, only of i_{j+1}. Eve has to collect an offer from the original shop i_j for her chosen j in order to maintain the chaining relation's validity at $j - 1$. Then Eve does the following:

$$\begin{aligned}
E \rightarrow i_j &: \mathit{\Pi}_E,\ \{\mathcal{M}_0,\ \ldots,\ \mathcal{M}_{j-1}\} \\
i_j \rightarrow i_{j+1} &: \mathit{\Pi}_E,\ \{\mathcal{M}_0,\ \ldots,\ \mathcal{M}_j\} \\
i_{j+1} \rightarrow E &: \mathit{\Pi}_E,\ \{\mathcal{M}_0,\ \ldots,\ \mathcal{M}_{j+1}\}
\end{aligned}$$

Upon the agent's return, Eve throws away $\mathcal{M}_{j+1}$, increments j, and picks a new i_{j+1}. The chaining relation and encapsulated offers are build as if Alice's agent had requested the offer (instead of Eve's agent with Eve's *program*) because $\mathcal{M}_0$ bears Alice's signature. Eve repeats the process at her discretion. When she is finally satisfied with the collection of encapsulated offers she assembled, she pastes them into Alice's agent, and sends that agent to i_{j+1}. If Alice can be fooled into forwarding agents whose $\mathcal{M}_0$ she signed herself then Eve's charade can carry on until the very last (faked) hop. Otherwise, Eve has to stop her attack before the next to last hop.

It must be stressed here that the problem is *not* that Eve can truncate offers and grow a fake stem (this possibility is acknowledged by the authors, so this fact is not surprising). The problem is that shops can be abused as oracles for generating offers to the terms of Eve rather than Alice (this remark also holds for Sects. 3.2 and 3.3). In other words, Alice might look for blue or red shirts with a preference on blue ones; she might find out that Eve is the only shop that offers her blue shirts, though. This is possible because Eve's agent looks only for red shirts, and the offers made to this agent are returned to Alice.

3.2 Chained Digital Signatures with Forward Privacy

The second protocol proposed in [5] is the *chained digital signature protocol with forward privacy* (P2). It is a variation of the protocol discussed in Sect. 3.1, with the order of encryption and signature computation being swapped. The goal of this arrangement is to hide the identity of shops that provided offers while keeping the integrity assurances. The protocol is defined as given below:

$$\begin{aligned} \mathcal{M}_n &= \{\{m_n\}_{S_{i_n}^{-1}}, r_n\}_{K_A}, C_n \\ C_n &= h(\mathcal{M}_{n-1}, r_n, i_{n+1}) \end{aligned}$$

$$i_n \rightarrow i_{n+1} \ : \ \mathit{\Pi},\ \{\mathcal{M}_0,\ \ldots,\ \mathcal{M}_n\}$$

A problem we couldn't resolve is how a shop knows who the owner of an agent is, and hence for whom the offers must be encrypted. The shop cannot extract the identity of Alice from $\mathcal{M}_0$, because the signature of the dummy offer m_0 is hidden by the encipherment. The authors leave that to speculation. The protocol's description is far from being sufficiently detailed at this point. Whereas a signer's identity can be verified easily against her signature using a public key and corresponding certificate (where the identity binding is assured by a certification authority), anybody could have used somebody else's *public* key to encrypt data.

Again, we assume that Eve received Alice's agent, and Eve is i_n as in the previous attacks. Let j be the smallest number for which Eve knows i_j. Eve probably knows i_{n-1} because this is most certainly the host that sent her the agent. In any case she knows i_n (her own identity).

Eve collects arbitrary signed offers using agents of her own, including an offer from i_j. Then, she cuts off the chain at j, and appends the offers, starting with the fresh one collected from i_j and the remaining ones in arbitrary order. In doing so, she generates random nonces as required, and builds the chaining relations consecutively from known data. The last chaining relation is computed with the identity of the entity to whom Eve wants to hand off Alice's agent.

Upon the agent's return, Alice cannot decide whether her agent remained unattacked, or carries offers of shops it has never seen actually. It is worth noting that the integrity assurance of the protocol relies on the secrecy of the association of $\mathcal{M}_j$ with the identity of the shop who signed offer m_j. This means that privacy of offers is not only a *feature* of the protocol, but is also a *requirement*. In particular, secrecy of the agent's itinerary is a requirement.

Once again, not the truncation of protocol data is the important point, but Eve's ability to set the terms for (authentic) offers returned to Alice.

3.3 Publicly Verifiable Chained Signatures

Another protocol that is proposed in [5] is the *publicly verifiable chained signatures* (P4) protocol. The key aspect of the protocol is that each shop generates a temporary asymmetric key pair (either on the fly or by means of precomputation) to be used by the successor. The public key is certified by the shop that generated the key pair. Each shop uses the private key that it received from its predecessor for signing its partial result, the chaining relation, and the public key to be used by its successor. The private key is destroyed subsequently. Let (χ_A, χ_A^{-1}) be a temporary key pair generated by A. The protocol is as follows:

$$\begin{aligned} \mathcal{M}_n &= \{\{m_n, r_n\}_{K_A}, C_n, \overbrace{\chi_{i_n}}^{\text{oracle}}\}_{\chi^{-1}_{i_{n-1}}} \\ C_n &= h(\mathcal{M}_{n-1}, i_{n+1}) \end{aligned}$$

$$i_n \rightarrow i_{n+1} \; : \; \mathit{\Pi},\ \{\mathcal{M}_0,\ \ldots,\ \mathcal{M}_n\}, \underbrace{\{\chi^{-1}_{i_n}\}_{K_{i_n, i_{n+1}}}}_{\text{oracle}}$$

The protocol is initialized by Alice with:

$$\begin{aligned} \mathcal{M}_0 &= \{\{m_0, r_0\}_{K_A}, C_0, \chi_A\}_{S_A^{-1}} \\ C_0 &= h(r_0, i_1) \end{aligned}$$

It is easy to see that Eve can collect valid certified temporary key pairs from Bob, simply by dispatching and agent of her own to Bob, which promptly returns to Eve. On the agent's transport to Eve, Bob sends a temporary private key χ_B^{-1} and corresponding certified public key χ_B (contained in $\mathcal{M}$).

We assume that Eve is i_n and she received Alice's agent. Let j be the smallest number for which Eve knows $\chi_{i_j}^{-1}$. She received $\chi_{i_{n-1}}^{-1}$ with the agent, so at least one such j exists and $j < n$. Eve then cuts off all encapsulated offers following $\mathcal{M}_j$, and collects key pairs from all the shops in whose names she wants to fake offers, including shop i_{j+1}. Starting with i_{j+1}, she appends arbitrary offers, building the protocol data consecutively. The identity that Eve uses in the final chaining relation is the one of the entity to whom she wants to hand off Alice's agent (for instance Alice herself).

4 Protocols Using Secure Co-Processors

In [6], Karjoth proposes use of trusted secure co-processors as a means to protect mobile agents in a distributed marketplace. The setting equals that described in Sect. 3, with the exception that each shop i_n has a trusted tamper-proof hardware T_n (in brief, its *device*), which is issued and certified by a central market authority $\Re$. The market authority acts as a trusted third party for merchants and customers. By assumption, the channel between a shop and its device is secure against active attacks. Each device has its own asymmetric key pair, and is capable of computing suitable asymmetric ciphers, symmetric ciphers, and message digests. Furthermore, each device has the public key of the market authority, and uses it to authenticate the public keys of other devices.

At the beginning of the protocol, Alice chooses a random $\mathcal{K}$ and sets $C_1 = h(\mathcal{K})$. The protocol continues as follows:

$$\begin{aligned} i_{n-1} \to i_n &: \mathit{\Pi}_A, \{\mathcal{K}, C_n\}_{K_{T_n}}, \{\mathcal{M}_1, \ldots, \mathcal{M}_{n-1}\}, \\ &\quad \{C_1, \ldots, C_{n-1}\} \end{aligned}$$

$$\begin{aligned} i_n \to T_n &: \{\mathcal{K}, C_n\}_{K_{T_n}}, \{m_n\}_{S_{i_n}^{-1}}, \{K_{T_{n+1}}\}_{S_{\Re}^{-1}} \\ T_n &: \mathcal{M}_n = \{\{m_n\}_{S_{i_n}^{-1}}\}_{\mathcal{K}},\ C_{n+1} = h(\mathcal{M}_n, C_n) \\ T_n \to i_n &: \{\mathcal{K}, C_{n+1}\}_{K_{T_{n+1}}}, \{C_{n+1}\}_{\mathcal{K}}, C_n, \mathcal{M}_n \end{aligned}$$

$$\begin{aligned} i_n \to i_{n+1} &: \mathit{\Pi}_A, \{\mathcal{K}, C_{n+1}\}_{K_{T_{n+1}}}, \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \\ &\quad \{C_1, \ldots, C_n\} \end{aligned}$$

In the final protocol step, the last shop sends Alice the agent and the final checksum, which is encrypted with $\mathcal{K}$:

$$i_n \to i_0 : \mathit{\Pi}_A, \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \{C_1, \ldots, C_n\}, \{C_{n+1}\}_{\mathcal{K}}$$

Alice knows $\mathcal{K}$, so she decrypts $\{C_{n+1}\}_{\mathcal{K}}$, verifies the checksums consecutively from C_1 to C_{n+1}, decrypts $\mathcal{M}_1, \ldots, \mathcal{M}_n$, and finally she verifies the signatures.

We assume that Eve runs a shop in the electronic marketplace, which implies that she has a device certified by the market authority. Consider that Eve received an agent owned by Alice, so Eve is i_n. Eve now has a number of encrypted offers, an equal number of checksums, and $\{\mathcal{K}, C_n\}_{K_{T_n}}$, which can be decrypted only by her device.

From the protocol, we know that $C_{n+1} = h(\mathcal{M}_n, C_n)$. There is nothing secret about h, so in fact Eve can take j of the $n-1$ encrypted offers, shuffle them, and re-compute the appropriate checksums herself, beginning with the initial checksum C_1 (without ever going through her device). However, Alice expects to receive a matching $\{C_{j+1}\}_{\mathcal{K}}$ with her agent. Eve cannot encrypt her final checksum with $\mathcal{K}$ because she does not know it – but her device can do it for her! All Eve has to do is passing C_{j+1} in the place where her device expects to receive Eve's signed offer:

$$E \to T_n : \{\mathcal{K}, C_n\}_{K_{T_n}}, \underbrace{C_{j+1},}_{\text{substituted for Eve's offer}} \{K_{T_n}\}_{S_{\mathfrak{R}}^{-1}}$$

The device first extracts Alice's secret key $\mathcal{K}$ from $\{\mathcal{K}, C_n\}_{K_{T_n}}$, which is encrypted with the device's public key. Then the device uses $\mathcal{K}$ to encrypt what it thinks is Eve's signed offer. Only that it is not the signed offer but the checksum that must be passed back to Alice with her agent.

$$T_n : \mathcal{M}_n = \underbrace{\{C_{j+1}\}_{\mathcal{K}},}_{\text{oracle computation}} \; C' = h(\{C_{j+1}\}_{\mathcal{K}}, C_{j+1})$$

Eve also passed her own device's public key rather than that of another shop's device. What Eve gets back from her device is:

$$T_n \to E : \{\mathcal{K}, C_{n+1}\}_{K_{T_n}}, \{C'\}_{\mathcal{K}}, C_n, \underbrace{\{C_{j+1}\}_{\mathcal{K}}}_{\text{leaked result}}$$

In other words, given a set of signed offers $\mathcal{M}_1, \ldots, \mathcal{M}_j$ (which are encrypted with Alice's secret key $\mathcal{K}$), Eve can construct a valid representation of Alice's agent, and return it to Alice in a way that is indistinguishable from an ordinary run of the agent.

Eve can also collect signed offers herself (at her own terms) using agents of her own. For instance, let $\{m_B\}_{S_B^{-1}}$ be such an offer, collected from Bob. Eve sends this offer to her device, rather than one of her own offers:

$$E \to T_n : \{\mathcal{K}, C_n\}_{K_{T_n}}, \underbrace{\{m_B\}_{S_B^{-1}},}_{\text{Bob's offer}} \{K_{T_n}\}_{S_{\mathfrak{R}}^{-1}}$$

$$T_n \rightarrow i_n : \{\mathcal{K}, C_{n+1}\}_{K_{T_n}}, \{C_{n+1}\}_{\mathcal{K}}, C_n, \underbrace{\mathcal{M}_B}_{\text{Bob's offer encrypted with } \mathcal{K}}$$

The device returns the offer encrypted with $\mathcal{K}$. Offers prepared in this way can also be used by Eve in her attack on the checksum.

If Eve just wants to append offers that she collected to Alice's agent (following $\mathcal{M}_{n-1}$), then the attack is even simpler. All Eve has to do is passing her own device's public key to her device rather than that of another shop's device until she wants to hand off Alice's agent. In that case she either passes the public key of the next shop's device, or returns the agent to Alice herself.

In summary, Eve can delete and rearrange any offers brought by the agent, and insert forged offers collected by her, at any position[3] in the chain of results. This means in particular that the protocol does not achieve *forward integrity* as is claimed by its author. The surprising fact is that although secure co-processors are used, the protocol fails where some software only approaches succeed (for instance the *chained MAC protocol* [5]). The lesson that is to be learned is that tamper-proof hardware is no guarantee for improved security.

In order to prevent the attack on the final encrypted checksum, the device has to verify that the data input as the signed offer is "well-formed", in other words, actually constitutes a signature rather than random data. Providing typed driver APIs is not sufficient since the driver software itself can be tampered with (which exposes the device's raw hardware interface).

5 Authentication to the Rescue?

It might be argued that mutual authentication of hosts in the course of agent hand-off may inhibit some of the attacks we described. Upon closer inspection, it turns out that actually only one protocol of the ones we discussed may profit from this (although that protocol still remains vulnerable to some extent).

The *target state* (Sect. 2.1) does not profit for obvious reasons. The *append-only container* (Sect. 2.2) defines the crucial checksum C_n in a way that makes it impossible for a hop to verify intermediate targeted states. Consequently, Eve can arrange a targeted state in her attack at will, and there is no point for hop i_{j+1} to verify e.g., that the sender of the agent actually inserted element j. Neither does the *multi-hops protocol* (Sect. 2.3) benefit from authentication. Eve may always sign μ_{j-1} (the last element of $\mathcal{P}$) herself, replace the last element of $\mathcal{M}$ with her own identity, and complete her attack without raising suspicion.

[3] In general, Eve knows only $\{\mathcal{K}, C_n\}_{K_{T_n}}$, so if she touches any encrypted offers before n then she has to hand off the agent herself to Alice, and cannot let another shop do this. However, she can pass on the agent if she knows that it will return to her before it hops back to Alice.

The protocols described in Sects. 3.2 and 4 obscure or encrypt all protocol data that is passed from one hop to the next. Again, there does not seem to be a hook to improve the protocol's security by verifying protocol data against authentication results. In protocol P4 (Sect. 3.3), hosts are exploited as key--generating oracles. Authentication results can hardly be connected with anything useful either, unless the protocol itself is modified.[4]

This leaves protocol P1 (Sect. 3.1). This protocol has two important properties. First, the data that is added by each host is randomized, and thus cannot be reliably *reproduced* by means of an oracle exploit. Second, the protocol builds a strong backward chain including the signature of the agent's previous host. Each host can verify this chain back to $\mathcal{M}_1$, starting with the last element in $\mathcal{M}$ whose signer must be the authenticated previous hop of the agent.[5] This makes it impossible for Eve to hide her traces completely, although she can still launch her attack in one sweep rather than multiple rounds. But her attack must start at her own position in the existing chain, and she must appear as well at the end of her faked sub-chain, because she needs to hand off Alice's agent and pass the combined authentication and signature check as well.

6 Conclusions

One problem repeatedly occurred in the protocols we analyzed: a legitimate host could be abused by malicious hosts as an oracle that decrypts, signs, or otherwise computes protocol data on behalf of an adversary. These flaws could have been avoided, had the authors of the protocols taken the advice of Needham and Anderson [1] faithfully: "be careful, especially when signing or decrypting data, not to let yourself be used as an oracle by the opponent."

Mobile agent systems are particularly vulnerable to this type of attack because they are meant to work autonomously, and no human intervention is expected to happen in order to validate and authorize the processing of agents by cryptographic protocols. Hence, agent servers and agent owners must have means to decide whether protocol data that an agent requests to process or returns, actually belongs to that agent. This brings us to another of Needham's and Anderson's rules of good practice: "where the identity of a principal is essential to the meaning of a message, it should be mentioned explicitly in that message."

None of the protocols that involved signing as a means of authenticating protocol data actually signed a data type or recipient identity along with the

[4] Each host may certify its temporary key with an authenticated attribute that includes the identity of the agent's previous hop. However, in that case Eve simply sends her key-collecting agent first to the hop whose identity shall be certified by her next target, then to her target, and back to her.

[5] Due to an unfortunate choice of C_0, only Alice can fully verify the chain at 0.

data. Hence, protocol data that was collected by one entity appeared valid to other entities as well. Obviously, inclusion of a recipient's identity is not even enough, because protocol data from one agent instance can be used again in an attack on other agent instances owned by the same entity. Since mobile agents may be under way for a period of time that is hard to anticipate in advance, it is difficult to have a notion of "freshness". If this were not enough, the protocols also have to cope with multiple agents that run concurrently. Both, agent owners and legitimate hosts must therefore "be sure to distinguish different protocol runs from each other."

Each agent instance certainly constitutes a different protocol run. On the other hand, digital signatures affixed to an agent's code are not sufficient to distinguish one agent instance from another. This leads to the important conclusion that *digitally signing a mobile agent's code alone is not sufficient to assert agent ownership*.

However, this approach is a favorable one among contemporary mobile agent systems. A signature on code can be copied just like the code itself. Code is written to be re-used, so the agent *instance* is what renders an agent (a protocol run) distinct. Seen in this light, it is even less desirable to sign credentials that contain a *code base* rather than the code itself (as described e.g., in [3]), because this gives an adversary potentially more valid agent programs to choose from. Each agent program that is available from a particular code base can be used in conjunction with credentials that refer to the code base.

Instead, the owner of some agent should sign a *static kernel*, which includes the agent's code as well as enough redundancy to distinguish between two instances of the same agent. A cryptographic hash value of the kernel's signature may serve as a unique "anchor" to which protocol data can be bound by means of a digital signature.

Agent developers must still be aware of the fact that "a migrating agent can become malicious by virtue of its state getting corrupted" [10]. We cannot assume that a mobile agent properly represents the intentions of its owner, because – subsequent to its first hop – an agent's state is a function of its own program and state, and the state and program of the hosts that it visited.

Hence, any attempt to protect a free-roaming agent against interleaving attacks is probably futile unless the agent's code is carefully designed, such that it does not leak confidential data, and does not enter negotiations based on parameters stored in its mutable state.

7 Acknowledgments

This paper is a revised and extended version of [11], with a discussion of defensive measures omitted. We thank Günter Karjoth for his comments on the initial manuscript, which helped to improve its precision and focus, as well as for the inspiring discussion that evolved from the commentary. We also thank the anonymous reviewers for their comments, which, again, led to several improvements and extensions.

References

1. R. Anderson and R. Needham, “Programming Satan’s computer,” in *Computer Science Today*, vol. 1000 of *Lecture Notes in Computer Science*, pp. 426–441, Springer Verlag, 1995.
2. M. Abadi and R. Needham, “Prudent engineering practice for cryptographic protocols,” SRC Research Report 125, Digital Equipment Corporation, June 1994.
3. N. M. Karnik and A. R. Tripathi, “Security in the Ajanta mobile agent system,” Technical Report TR-5-99, University of Minnesota, Minneapolis, MN 55455, U. S. A., May 1999.
4. A. Corradi, R. Montanari, and C. Stefanelli, “Mobile agents protection in the Internet environment,” in *The 23rd Annual International Computer Software and Applications Conference (COMPSAC ’99)*, pp. 80–85, 1999.
5. G. Karjoth, N. Asokan, and C. Gülcü, “Protecting the computation results of free–roaming agents,” in *Proceedings of the Second International Workshop on Mobile Agents (MA ’98)* (K. Rothermel and F. Hohl, eds.), vol. 1477 of *Lecture Notes in Computer Science*, pp. 195–207, Berlin Heidelberg: Springer Verlag, September 1998.
6. G. Karjoth, “Secure mobile agent-based merchant brokering in distributed marketplaces,” in *Proc. ASA/MA 2000* (D. Kotz and F. Mattern, eds.), vol. 1882 of *Lecture Notes in Computer Science*, pp. 44–56, Berlin Heidelberg: Springer Verlag, 2000.
7. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Discrete Mathematics and its Applications, New York: CRC Press, 1996. ISBN 0-8493-8523-7.
8. V. Roth and V. Conan, “Encrypting Java Archives and its application to mobile agent security,” in *Agent Mediated Electronic Commerce: A European Perspective* (F. Dignum and C. Sierra, eds.), vol. 1991 of *Lecture Notes in Artifical Intelligence*, pp. 232–244, Berlin: Springer Verlag, 2001.
9. B. S. Yee, “A sanctuary for mobile agents,” in *Secure Internet Programming*, vol. 1603 of *Lecture Notes in Computer Science*, pp. 261–273, New York, NY, USA: Springer-Verlag Inc., 1999.
10. W. M. Farmer, J. D. Guttman, and V. Swarup, “Security for mobile agents: Issues and requirements,” in *Proceedings of the National Information Systems Security Conference (NISSC 96)*, pp. 591–597, October 1996.
11. V. Roth, “Programming Satan’s agents,” in *Proc 1st International Workshop on Secure Mobile Multi-Agent Systems*, (Montreal, Canada), May 2001. Available at URL `http://www.dfki.de/~kuf/semas/semas-2001/`.