Solving Fair Exchange with Mobile Agents

Henning Pagnia[1], Holger Vogt[1*], Felix C. Gärtner[1*], and Uwe G. Wilhelm[2]

[1] Computer Science Dept., Darmstadt Univ. of Technology, D-64283 Darmstadt, Germany
{pagnia|holgervo|felix}@informatik.tu-darmstadt.de
[2] Operating Systems Laboratory, Communication Systems Department, Swiss Federal Institute of Technology, 1015 Lausanne, Switzerland
Uwe.Wilhelm@epfl.ch

Abstract. Mobile agents have been advocated to support electronic commerce over the Internet. While being a promising paradigm, many intricate problems need to be solved to make this vision reality. The problem of *fair exchange* between two agents is one such fundamental problem. Informally speaking, this means to exchange two electronic items in such a way that neither agent suffers a disadvantage. We study the problem of fair exchange in the mobile agent paradigm. We show that while existing protocols for fair exchange can be substantially simplified in the context of mobile agents, there are still many problems related to security which remain difficult to solve. We propose three increasingly flexible solutions to the fair exchange problem and show how to implement them using existing agent technology. The basis for ensuring the security properties of fair exchange is a tamper-proof hardware device called a trusted processing environment.

1 Introduction

Ever since their emergence, mobile agents have been advocated to revolutionize the way in which we go shopping on the Internet. Instead of having to surf endlessly through the World Wide Web (WWW) and digesting huge amounts of possibly untrustworthy information, personalized mobile agents autonomously gather information about items we want to buy. Once they have found the WWW page, the piece of software or the latest hit single which we were looking for, the agent can perform the purchase on behalf of the user and – to our enjoyment – carry the item back home.

While being a promising paradigm, many technical aspects must be solved to make this vision reality. First of all, there are substantial security issues involved. For example, vendors which let mobile agents visit their WWW servers are usually not willing to execute arbitrary (possibly malicious) agent code on their local machine. Solutions to this problem have been proposed and have even been introduced into modern programming languages like Java. However, the converse of protecting the host from the agent is also a problem: agents often carry precious data such as personal information, electronic money, or (minor) company secrets. It must be ensured that such items cannot be stolen by other agents or robbed by visited hosts, and that agents cannot be kidnapped. From the existing solutions [6, 9, 11, 16] we have chosen to pursue one which is based

* This author's work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the two PhD programs (Graduiertenkollegs) "Enabling Technologies for Electronic Commerce" and "ISIA" at Darmstadt University of Technology.

on trusted hardware, such as the trusted processing environment (TPE) described by Wilhelm *et al.* [16].

However, the ability to protect the agent from fellow agents or hosts alone does not solve the manyfold problems associated with electronic business transactions on the WWW. For many transactions it is still necessary to provide mechanisms for executing an exchange of digital items in a fair manner. Intuitively, fairness means that either both participants receive their desired item or (in case of an unsuccessful exchange) nobody wins or loses something valuable. A naive protocol to perform such an exchange could be the following: an agent wishing to buy the soundtrack of "My Fair Lady" from a music vendor could proceed to the vendor's host, receive the audio file, pay for it electronically and travel back home again. There are two obvious problems with this simple protocol: 1) the agent could leave the vendor's host before paying, and 2) the vendor could hinder the agent from leaving the host after having paid. Obviously, both outcomes of the transaction are unfair.

The situation can be improved if the host contains or is connected to a TPE which is trusted. In such a scenario, the vendor places its own local agent onto the TPE and then both customer and vendor agent start to exchange their items fairly. But how to proceed? Assume for example, that the vendor agent first delivers the data file to the customer agent. Who can now prevent the customer agent from leaving the TPE? On the other hand, if the customer first hands the money to the vendor, who can guarantee that the vendor will readily hand over the data file? If a data file is returned, who guarantees that it is in fact the soundtrack of "My Fair Lady"? And how is it possible for customer and vendor to check the integrity of the exchanged item without revealing it to the other? The core problem encountered here is usually called the problem of *fair exchange*.

Due to its importance in electronic commerce, the problem of fair exchange has received increasing attention in recent years and a number of papers have been published in this area (see for example the papers by Asokan *et al.* [2], Franklin and Reiter [4] or Vogt *et al.* [14]). In almost all of these papers, the problem has been studied in a general message-passing network scenario which makes solutions to the problem rather complex (if not impossible [8]). This results from the possible occurrence of communication faults or malicious process behavior (e.g., intentionally stopping to proceed in the protocol or sending corrupted messages). We will show that protocols to solve fair exchange can be substantially simplified when adapting them to the agent scenario. To our knowledge, fair exchange in this context has not been addressed in the literature yet.

In this paper, we consider the following scenario: two agents, A and B, meet on a trusted host (implemented by a TPE) to exchange two items in a fair manner. We propose three novel solutions to the fair exchange problem in the mobile agent environment which are both practical and efficient, we sketch how they can be implemented in existing agent platforms, and we discuss their relative merits and shortcomings.

We proceed as follows: In Section 2 we introduce the concept of a TPE which is the basic building block of our solutions. In Section 3 we define the fair exchange problem and state why the mobile agent scenario bears the potential of simple solutions to a difficult problem. We then present these solutions in Sections 4 and 5. Finally, we summarize and discuss our results in Section 6.

2 Trusted Processing Environment

The necessity to protect mobile agents from their execution environment has been addressed in several publications [6, 9, 11, 16]. Here, different solutions have been proposed for the problem of protecting the data of a mobile agent from undesired disclosure and its code from undesired interference. We have chosen to pursue the approach identified by Wilhelm *et al.* in [16], which relies on the usage of trusted and tamper-resistant hardware. This approach proposes to formalize the guarantees into a policy that is associated with the underlying hardware. Such a policy consists of a set of rules prescribing the behavior of the hardware for all relevant situations. A similar protection and the enforcement of a meaningful policy are either very difficult or impossible to achieve in any of the other mentioned approaches. We will first explain the overall approach by introducing the trusted processing environment (TPE) and then explain some of the policies that can be enforced by a TPE.

2.1 Trusted and Tamper-resistant Hardware

As noted above, the basic idea of the TPE is to rely on trusted and tamper-resistant hardware to protect a mobile agent from interference by the owner of the TPE. This implies that we have to trust the manufacturer of the TPE to properly design and produce its TPEs, which is a basic assumption underlying the entire approach (a more extensive discussion of this assumption can be found in [16]). Thus, we assume that the host computer which is under the control of the TPE owner, cannot interfere with the task of the TPE other than through a restricted interface. This interface is completely controlled by the TPE and allows, for instance, to upload and remove agents or to interact with agents on the TPE. The TPE (see Figure 1) is a complete computer and provides an agent execution environment which cannot be inspected or tampered with. It is designed as a closed system that contains its entire operating code (i.e., operating system, virtual machine for mobile agents, I/O interface for external interactions, and cryptographic library) in non-volatile protected memory (e.g., internal ROM) that allows a secure boot of the system. Finally, the TPE contains the private key K_{TPE}^{-1} of a cryptographic key pair (K_{TPE}, K_{TPE}^{-1}). K_{TPE}^{-1} is generated by the TPE and is known to no principal other than the TPE itself – including the TPE's physical owner. The secrecy of this private key, ensured by the tamper-resistance of the TPE, is a crucial requirement for TPE usage to enforce a particular behavior. The actual guarantees towards the physical protection offered by the TPE are the following:

1. The TPE is tamper-resistant and its private key has not been compromised.
2. The virtual machine inside the TPE correctly implements the specification of the programming language used by the mobile agent.
3. The cryptographic library implements the cryptographic algorithms correctly.
4. Any access to or manipulation of data on the TPE is under the control of the operating system.
5. The operating system protects a mobile agent from other agents on the same TPE.

The protection offered by the TPE and its conformance to a particular policy is ensured by the manufacturer, which provides the TPE owner with an appropriate certificate. This certificate contains information about the TPE, such as manufacturer, type, the provided guarantees, the enforced policy, and the public key (K_{TPE}). Thus, in order to securely

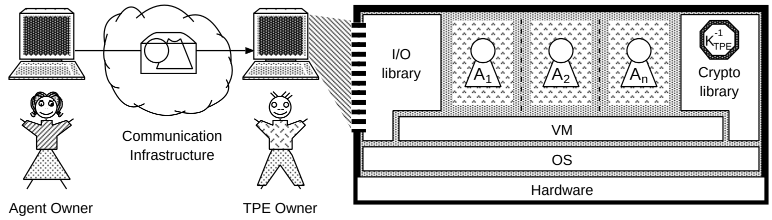

Fig. 1. Mobile agents on the trusted processing environment (TPE)

send a mobile agent to a particular TPE, in principle the following steps have to be taken. The sender has to obtain the certificate of the TPE, verify the policy enforced by the TPE, assess the assurance in the tamper-resistance of the TPE, and send the mobile agent encrypted with the public key of the TPE to the TPE owner. This guarantees that the information within the mobile agent cannot be accessed or tampered with while the mobile agent is in transit. Once the mobile agent has arrived on the TPE, it is protected by the tamper-resistance of the TPE and the policy implemented in the TPE as described next.

2.2 Using the TPE

As we have previously stated, the presented approach was primarily conceived to provide a well defined protection to a mobile agent. A detailed discussion of the many possible protection goals and the corresponding policies is outside the scope of this presentation (for further information see [15]). We will only discuss the policy rules that are most relevant to the problem of fair exchange and point out how mobile agents can take advantage of them. We now assume a TPE that enforces the following rules:

(R1) The code of an agent will never be disclosed or altered by the TPE and any invocation of the agent's methods will be executed exactly according to the code of the agent.

(R2) The data of an agent can exclusively be accessed and manipulated through the interface of the agent. If the agent does not provide methods to directly access a particular data item, its value can at most be inferred from the responses to other method invocations.

(R3) The TPE will protect the mobile agent from any interference from other agents on the same TPE (other than calls to the mobile agent's public interface).

(R4) Prior to a migration, an agent will obtain the certificate of the designated receiver's TPE. The agent can decide whether it wants to be transferred and the current TPE will honor the agent's decision. For the transfer, the agent will be encrypted with the public key of the receiver's TPE.

(R5) The TPE establishes a unique identifier for each mobile agent on the TPE and provides an invokee with the ID of the invoker. The TPE invokes a method exclusively and unaltered on the mobile agent with the ID provided in the invocation.

(R6) The TPE provides an operation that, given the ID of a mobile agent A, returns the cryptographic hash of the immutable part of A. The TPE will query A if it approves of this operation.

(R7) The TPE guarantees the survival of a mobile agent despite the possibility of a temporary hardware failure.

This policy allows a mobile agent, for instance, to rely on its own code in order to hide parts of its state (e.g., cryptographic keys), to control its migration to other TPEs, and to ensure the confidentiality, integrity, and authenticity of local method invocations. Provided that the policy of the TPE will be enforced (which is ensured by the manufacturer), it is quite obvious how the mobile agent can implement these protection goals. A very interesting protection goal in the context of fair exchange can be realized based on rule R6, which allows a mobile agent to determine whether it interacts with an agent executing some specific code. This mechanism is called *authentic code* [15]. The underlying idea is that the implementation of a mobile agent can be analyzed in order to ensure that it correctly implements a well-defined behavior.[1] An owner will then configure his mobile agent to only interact with a mobile agent that is identical to the one he has previously analyzed. This allows a mobile agent, for instance, to disclose a particular piece of information exclusively to an agent with a completely known behavior. We will discuss this issue in more detail in Section 5.3.

3 The Fair Exchange Problem in the Mobile Agent Scenario

While the notion of fairness in electronic commerce feels rather simple to understand, it is quite difficult to formalize [5]. In the context of a fair exchange, fairness intuitively means that either both participants receive their desired item or (in case of an unsuccessful exchange) nobody wins or loses something valuable. This means that if, for example, Alice and Bob want to exchange items, a system must *not* allow, for example, that Bob can obtain access to Alice's item while Alice cannot access Bob's item. More specifically, a system that implements a *fair exchange protocol* between Alice and Bob must ensure three conditions [1, p. 9]:

1. Effectiveness: If Alice and Bob behave correctly and if the protocol terminates, then Alice will have Bob's item and vice versa.
2. Timeliness: The protocol will eventually terminate.
3. Fairness (see below).

We will now discuss the third condition in more detail. Fair exchange protocols must particularly protect against the situation in which either party refuses to cooperate in order to gain an advantage. Existing fair exchange protocols are rather complicated due to the asynchronous nature of the underlying network: problems occur because in this system model communication can be interrupted, or nodes can crash. Hence, a proper termination of the exchange process cannot be guaranteed. In order to overcome this restriction different notions of fairness have been introduced [1]:

a) *Strong fairness*: When the protocol has completed, either Alice has Bob's item or Bob has gained no additional information about Alice's item, and vice versa [1, p. 9]. In other words: A participant who behaves correctly does not suffer a disadvantage, i.e., it is never the case that Alice receives Bob's item but Bob does not receive Alice's item, or vice versa.
b) *Weak fairness*: Either strong fairness is achieved, or a correctly behaving participant Alice can prove to an arbiter that Bob has received (or still can receive) Alice's

[1] Of course, the effort to analyze an agent can be very high. However, since such an agent must only be analyzed once and can then be used *many* times, the effort is often justified.

item without any further intervention from Alice [1, p. 9] (no matter whether Bob behaves correctly or not), and vice versa.

Strong fairness is achieved on protocol completion usually by using a trusted third party which first collects the items to be exchanged and then (after checking their validity) swaps them. Weak fairness is reached if either the exchange process terminates in the way as described above or both parties collected sufficient proofs in order to enforce the exchange after protocol completion with the help of an outside arbiter.

It can be shown that it is impossible to reach strong fairness in an asynchronous system without a trusted third party [8]. It turns out that even by involving a trusted third party (TTP) strong fairness cannot be guaranteed in an asynchronous system without making special assumptions. Assume that the TTP has collected the items from the two exchange parties Alice and Bob. Next, the TTP forwards Bob's item to Alice but crashes before it sends the other item to Bob. (Alternatively, one can assume that the communication link to Bob breaks down.) Now, Alice has received the item but Bob has not. Hence, the exchange did not terminate in a fair way. Fair exchange protocols therefore are implemented with reliable communication and persistent TTPs. Exploiting these properties, they define strong fairness to be reached if both parties *eventually* receive the item [12].

As discussed in Section 2.2 a TPE must provide solutions to many of the problems which also have to be solved for a TTP, particularly reliability and persistency. This means, that the TPE must ensure that agents cannot die due to node crashes [7] and that agents migrate in a reliable "exactly-once" fashion from host to host [10]. For performing fair exchange, we let all three parties involved (TTP, Alice, and Bob) reside on the same node. In this scenario neither can communication break down nor can a single party crash. If the TPE node crashes before fair exchange was committed, then all parties are destroyed, thereby resetting the exchange process. If it crashes after the exchange has been performed then the TPE must ensure that the agents are correctly recovered. Because of these properties the mobile agent scenario is so well-suited for solving fair exchange.

Still, for items which might expire or items whose value decreases by time (e.g., stock quotes or time table information) the exchange is problematic: One party might receive the item in time whereas the other party receives it after its expiration, rendering the item useless. Thus, the so achieved degree of fairness might still be unsatisfactory.

4 Fair Exchange Using Designated TPE Functionality

4.1 The Basic Protocol

We propose the use of mobile agents meeting at a special-purpose TPE. This TPE offers designated fair exchange functions and is thus extended by new methods which we introduce in Section 4.2. Alice sends its mobile agent A to the TPE while Bob sends agent B. Each agent carries the item which is intended to be exchanged and includes a routine for checking the expected item. When the agents meet, the TPE must ensure that they play fair, i.e., only if the items were exchanged successfully and both agents are content, the agents are allowed to move on. Furthermore, none of the agents must be allowed to communicate with any other party except for their exchange partner. This prevents that a party might gain an advantage by forwarding the newly received item to an outside party before the exchange is committed.

The protocol we propose is as follows: As shown in Figure 2 one can think of the two agents (A and B) to be locked in an empty room with a single door but no windows. After the agents have entered the room they are not released unless both signal that the exchange was successful. The agents will only do this if they have checked the exchanged items and are content with what they received. The exchange is assumed to be completed only if both have signaled a "Commit" message. Otherwise the door remains locked. If either agent decides "Abort" then both agents will be destroyed.

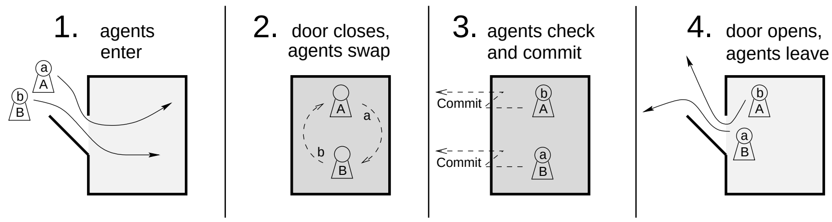

Fig. 2. Fair exchange in a locked room.

Defining a fair exchange protocol for this scenario is much simpler than in the distributed scenario. The reason for this is that the agents are isolated during the exchange process. Consequently, swapping the items does not need to take place atomically provided that in the meantime neither agent is capable of spreading information to the outside. Opening the door finally results in simultaneously releasing the two agents and effectively re-enables their ability to communicate. Now that the items have been exchanged, the agents should be retransmitted to their owner. This however, is no longer part of the fair exchange protocol but is under control of the agent platform. The exchange protocol guarantees that the agents swap the items in a fair manner. But it has not to deal with the way both agents take the received items back to their owner. The reason for this is that mobile agents are allowed to visit an arbitrary number of sites before returning to their owner and hence, an agent might have more tasks to fulfill after having executed the exchange. Therefore, the exchange process has to be regarded complete as soon as the items have been exchanged between the agents.

4.2 Implementing the Protocol Using a Designated TPE

The TPE offers fair exchange functionality through three publicly accessible methods:

- `BeginFairExchange(AgentId id)`
- `CommitFairExchange()`
- `AbortFairExchange()`

Figure 3 shows a possible implementation of these methods in pseudocode. Their semantics is as follows: Upon invocation of `BeginFairExchange(B)`, the TPE guarantees that the invoking agent can only invoke its own local methods as well as communication with agent `B`. Note that the agent that enters the room first is blocked until its partner has arrived. Each agent is then allowed to make one call of either `CommitFairExchange()` or `AbortFairExchange()`. These restrictions are maintained by the TPE until one of the following cases arises:

1. Both the invoking agent *and* agent `B` invoke `CommitFairExchange()`. In this case execution resumes normally.

2. At least one of the two agents invokes `AbortFairExchange()`. In this case both agents are terminated and a notification about this fact is sent to the parent of the agents (either the agent owner or the mobile agent that spawned the participating agent).

The solution is fair because there is no way to "smuggle" an item out of the room without the other party's consent. The corresponding security property that there is no flow of information from inside the room to the outside during the exchange process.

Instead of terminating both participating agents in case 2, the agents could as well be reset into the state they were in before invoking `BeginFairExchange()`. However, this involves technically complex mechanisms to ensure that agents do not enter an infinite loop. To us, it seems simpler to send out "one-time" agents to perform the fair exchange and destroy the agents in case the exchange fails.

Agent code of A:

```
⋮
BeginFairExchange(B);
send(B, itemA);
receive(B, itemB);
if (checkB(itemB) == OK)
   CommitFairExchange();
else
   AbortFairExchange();
endif;
⋮
```

Agent code of B:

```
⋮
BeginFairExchange(A);
send(A, itemB);
receive(A, itemA);
if (checkA(itemA) == OK)
   CommitFairExchange();
else
   AbortFairExchange();
endif;
⋮
```

Methods provided by the TPE:

```
method BeginFairExchange(AgentId you) is
begin
  d[me] := UNCERTAIN;
  if NOT(is_waiting(you))
    wait(me);
  else
    close_door;   -- block communication
    wakeup(you);
  endif;
end;

method CommitFairExchange() is
begin
  d[me] := COMMIT;
  if (d[you] == UNCERTAIN))
    wait(me);
  elseif (d[you] == COMMIT))
    open_door;  -- resume communication;
    wakeup(you);
  endif;
end;

method AbortFairExchange() is
begin
  d[me] := ABORT;
  destroy(you);
  destroy(me);
end;
```

Fig. 3. Possible implementation of the fair exchange policy.

5 Fair Exchange Without Extending Existing TPE Functionality

TPEs are complex devices which are not easy to manufacture and require a complex certification infrastructure. Therefore, adding specific pieces of hard- or software to the

TPE, as proposed in the previous section, might not always be a suitable solution. In this section we show that under certain assumptions the exchange functionality placed within the TPE can be incorporated into a specialized agent. This solution avoids the use of a designated special-purpose TPE but also has inherent problems because the exchange agent has to be flexible and trustworthy at the same time. In Section 5.1 we study these problems in more detail and discuss some solutions in Section 5.2. Subsequently, we present two increasingly flexible solutions for fair exchange in Sections 5.3 and 5.4.

5.1 The Check-Routine Problem

The simplest solutions to the fair exchange problem which do not alter TPE functionality are based on the concept of a *fair exchange agent* (FEA). A FEA is an agent that acts as a trusted intermediate to the two agents A and B which want to perform a fair exchange. If such an intermediate is available, the exchange can proceed as follows: A and B both send their item to the FEA. The FEA checks whether A has sent the item which B wants to have and vice versa. If both checks succeed, the FEA forwards A's item to B and B's item to A. This type of protocol is a *fair exchange protocol with an active trustee*.

In contrast to the solution presented in Section 4, the validity check of the items must occur within the FEA itself (and *not* in A and B) in order to ensure fairness. But note that because the items in consideration for exchange will usually be different ones in each instance of the problem, the activity of checking the items will be different every time the FEA is used. Thus, prior to executing fair exchange, both A and B must indicate to the FEA how it should check the other's item for validity.

The simplest solution would be for both A and B to devise a specific `check` method which takes the other's item as an input and returns `true` only if the item actually has the desired features. This approach is rather flexible because it enables to check an item in all ways allowed by the underlying programming language. For example, it would be possible to calculate cryptographic checksums over the entire item or over parts of it, it could compare parts of the item with some test data, or it could perform file format checks (e.g., "is the data a JPEG image?").

Subsequently, agent A devises a method `checkB` and B devises a method `checkA`. Both A and B send the code of this methods to the FEA before the fair exchange takes place. Within the fair exchange protocol, the FEA uses `checkB` to check the item sent by B and `checkA` to check the item sent by A. Only if both methods return `true` will the FEA complete the exchange.

The problem with this solution stems from its flexibility: If arbitrary code is allowed within the `check` routines, it is possible to cheat. To understand this, imagine that A devises a routine `checkB(item)` which first sends `item` to itself (i.e., to A) and then returns `false`. As a result, the exchange is in danger of ending in an unfair situation where A has obtained B's item but B has not received A's item. Consequently, it is necessary to guarantee that nothing "bad" happens within the `check` routines.

The bad things which may happen within the `check` routines are similar to those which were forbidden within the fair exchange policy described in Section 4: there must be no flow of information from inside the `check` routine to another agent. This should include information flow through hidden channels [3], which is particularly difficult.[2]

5.2 Solving the Check-Routine Problem

This section contains a discussion of four possible solutions for the check routine problem. It depends on the application scenario which one should be selected.

Parameterized `check` routines. One solution is to use a predefined check routine from a specialized library. For example, the FEA could provide a set of publicly accessible methods, e.g., to check whether a file represents a valid JPEG image. In this case an agent could simply send the identifier of the desired check routine to the FEA prior to the exchange.

Alternatively, the FEA could offer a generic check routine which is parameterized with a predefined range of values. This could be, for example, a routine which scans the input for a set of keywords (the parameter of the routine would be the set of words). Extending this idea, the generic check routine would take an expression in a formal "item description" language as an argument. A drawback of this approach is that this requires the definition of a rather complex language which is sufficiently expressive in order to allow the check for all relevant item properties.

Syntactic check. Instead of using generic `check` routines, A and B could still be allowed to write their own methods. In this case it must be possible to automatically verify that their respective routines play according to the rules. For example, by automatically scanning the agents' code before invoking it, it is possible to check whether a method contains the invocation of a `send` command. If no other way exists to smuggle information out of the `check` function, then fairness can be guaranteed. However, for leaking information usually other possibilities than using the `send` command exist. Fraudulent agents are likely to garble their harmful actions within innocent looking code and preventing agents from performing such malicious actions solely by using syntactic checks is a difficult task.

Mutual code checking. Instead of the FEA having to validate the `check` routines, this task can be transferred to the participating agents. This means that A validates B's `checkA` routine and vice versa. This solution makes the task easier for the FEA, but has its own limitations. Consider for example the following scenario: A is looking for a text containing a set of keywords and so it devises a method `checkB` which simply scans the item for these words. The code of this method is sent to B so that B can validate that nothing bad happens therein. But knowing the code of `checkB`, B can easily fool A by piecing together an arbitrary text containing these keywords. While not being unfair in a formal sense, the disclosure of its `check` routine lowers an agent's confidence in the quality of the received item.

The described scenario is by no means exotic: it always occurs when in addition to syntactic checking an agent wants to perform "semantic" checking by using heuristics.

[2] A problem which arises in the context of electronic money is that it is impossible to prevent double spending without online access to a bank. This means that the `check` routine *must* make an online query when checking the money. In this case we can only ensure that the amount of information flow out of the `check` routine is bounded.

Instances of this problem are, e.g., plausibility checks on a credit card number or simple spot checks on mass-produced articles. Disclosure of the `check` routines bears the potential of cheating as explained above.

Sandboxing. A more powerful solution for verifying that the check routine well-behaves is to monitor its execution. Any attempt to execute an unfair command will be recognized at run-time. It will result in an immediate termination and rejection of the check routine. This approach for protecting the participating parties from malicious code is comparable with the sandbox model for Java applets. In contrast to the syntactic check, sandboxing is more immune against innocent looking, obfuscated code which is intended to deceive the other party. Another advantage of this security concept is that the timely termination of the check routines can also be monitored. If the execution time exceeds a previously defined time limit, the FEA stops computation and aborts the exchange. A limitation of sandboxing is its limited flexibility: Any command must either be allowed or forbidden. If for example communication is forbidden, then the client has no means to, e.g., obtain certificates from a key server.

5.3 Using Authentic Code

In this section we propose a novel solution for fair exchange which is not based on a designated TPE but instead uses a trusted third agent. The trust in such an agent is based on rule R6 of Section 2.2: The agents A and B first cooperate to build a new agent (see below) and they both check whether this agent executes the exchange in a fair way. Then one of them starts this newly constructed agent, and the other party can query the cryptographic hash value of this agent. If it matches the hash of the previously checked agent, the started agent and the checked agent are both identical with overwhelming probability. This means that the started agent can be trusted to guarantee fairness.

The construction of this fair exchange agent consists of two parts: The basic functionality for the fair exchange is always the same and can be implemented by taking an appropriate piece of code out of a publicly available library (a pseudocode implementation is given in Figure 4). As all parts of such a public library usually have been examined in advance, they don't have to be checked again when they are used for the construction of the fair exchange agent. The second part of the fair exchange agent consists of the check routines. These are exchanged between the agents A and B and must then be carefully examined as described in the paragraph about mutual code checking in Section 5.2. Only if the check routines turn out to be fair, they are built into the fair exchange agent.

```
begin
  receive(A, itemA) || receive(B, itemB);
  if ((checkA(itemA) != OK) OR (checkB(itemB) != OK))
    send (A, ``ABORT'') || send (B, ``ABORT'');
  else
    send (A, itemB) || send (B, itemA);
  endif;
end;
```

Fig. 4. The pseudocode of a generic fair exchange agent.

The advantage of this approach is that the agents A and B are able to solve the fair exchange problem without further assistance. They can trust their newly constructed fair exchange agent because of the following two reasons: First, the code is known to both agents and has been checked for malicious instructions. Second, they can be sure that the fair exchange agent they use is identical with the previously examined code.

5.4 Using an Independent Fair Exchange Agent

The solution which uses authentic code for providing trust in the FEA has some drawbacks: The construction of the complete FEA might cause of lot of communication between the two agents A and B, and checking the fairness of the newly constructed agent might require a huge amount of computation. Apart from that, some properties (like the termination of the FEA) cannot be entirely validated and authentic code is not available on every platform for mobile agents.

A different solution for fair exchange uses a certified "freelance", i.e., independent FEA (IFEA), which acts as an active TTP: The IFEA is an agent which receives both items, checks them and performs the swap, if the items have the desired properties. In contrast to the solution with authentic code, the agents A and B needn't know the code of the IFEA in order to trust it. Instead, they have to trust the party which has created the IFEA. This trust can be established by a certificate which is issued by this party for its IFEA. This certificate ensures that the party has instructed the IFEA to perform an exchange in a fair way. The agents A and B must then only be able to verify the certificate provided with the IFEA.

When the agents A and B decide to start a fair exchange of their items, the IFEA is probably already running. Therefore it has to be very flexible in exchanging all kinds of items. This can be achieved if an agent which wants to exchange an item sends its code for item checking to the IFEA. This again leads to the problem of detecting malicious check routines which is discussed in detail in Section 5.1. In contrast to the solution with authentic code, the burden of solving the `check` routine problem (with one of the approaches from Section 5.2) is now placed on the IFEA.

In the faultless case both check routines will usually accept the items which have been delivered to the IFEA. The IFEA forwards the items to the agents A and B and finishes the exchange by performing a reset. Now the IFEA can again be used for another fair exchange.

6 Summary and Conclusions

Due to its omnipresence, the fair exchange problem is an important problem in electronic commerce. It is at the core of almost every application processing electronic business transactions and by no means a problem solved easily. Fair exchange protocols must be devised carefully in order to cope with the many fault-tolerance and security problems encountered in today's networks.

In this paper we have shown that protocols for fair exchange can be substantially simplified by adopting the mobile agent paradigm of electronic commerce. The simplifications stem from the fact that many of the fault-tolerance and security problems which are present in the general network case can be ignored in the mobile agent scenario. For example, in the mobile agents paradigm it is assumed that agents can roam the network in a fault-tolerant manner, that their cargo is protected from disclosure and that the environment has the power to reset or terminate agents violating accepted behavioral policies.

Table 1. Summary of the three solutions.

	Solution	Section	What is revealed in advance?	Trust ensured by?
1.	Change the TPE	4	nothing	TPE policy
2.	Authentic Code	5.3	check routine	FEA code analysis and TPE (through authentic code)
3.	Independent FEA	5.4	nothing	Trust in IFEA owner

The three solutions we have proposed are summarized in Table 1. Algorithmically, solution 1 is probably the simplest one, but it involves adding functionality to a TPE. This might be prohibitively expensive since TPEs are complex devices which must be certified. Solutions 2 and 3 build upon existing TPE functionality, but both involve solutions to the check routine problem described in Sections 5.1 and 5.2. Solution 2 places this burden upon the individual agents, i.e., they have to be able to analyze/monitor each other's `check` routine. This solution works because both parties can trust in the authenticity of the FEA, which is guaranteed by the authentic code policy of the TPE. Solution 3 relieves the participating agents from solving the check routine problem by utilizing specialized agents to perform this task. Consequently, this solution seems to be the most "agent-friendly" way to offer a fair exchange service over an existing TPE infrastructure. However, the specialized FEAs have to be certified, a cost which must not be underestimated. In contrast, solution 2 does not involve (additional) certification overhead.

A point which is of interest to researchers studying fair exchange in the asynchronous network case is the following: Because the agent execution environment solves many of the fault-tolerance and security problems at hand, our solutions only need to solve the remaining exchange and security functionality. In this sense, they crystallize the "essence" of these parts of the exchange problem. Encapsulation of this functionality in program modules could lead to a more modular way of building fair exchange protocols [14].

The solutions we have presented are both practical and efficient. They are practical because they can be implemented using existing or devised agent technology. They are efficient because all computations are local to one network host and solution 1 (Section 4) requires only two agent interactions (which is minimal). The efficiency of solutions 2 (Section 5.3) and 3 (Section 5.4) is only bounded by the amount of computation necessary to solve the `check` routine problem.

Still some questions remain: For example, despite the fact that we achieve strong fairness in our protocols, it is usually a user concern that a well-behaving participant can still prove that it has acted according to the protocol. How can this feature be added to our solutions? Future work will evaluate the applicability of our solutions in agent-based online shops.

Acknowledgments. We wish to thank Wolfgang Theilmann for his comments on a previous version of this paper and the anonymous referees for their useful feedback.

References

1. N. Asokan. *Fairness in electronic commerce*. PhD thesis, University of Waterloo, May 1998.
2. N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In T. Matsumoto, editor, *4th ACM Conference on Computer and Communications Security*, pages 8–17, Zurich, Switzerland, Apr. 1997. ACM Press.
3. D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.
4. M. K. Franklin and M. K. Reiter. Fair exchange with a semi-trusted third party. In T. Matsumoto, editor, *4th ACM Conference on Computer and Communications Security*, pages 1–5,7, Zurich, Switzerland, Apr. 1997. ACM Press.
5. F. C. Gärtner, H. Pagnia, and H. Vogt. Approaching a formal definition of fairness in electronic commerce. In *Proceedings of the International Workshop on Electronic Commerce (WELCOM'99)*, pages 354–359, Lausanne, Switzerland, Oct. 1999. IEEE Computer Society Press.
6. F. Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. In Vigna [13], pages 92–113.
7. D. Johansen, K. Marzullo, F. B. Schneider, K. Jacobsen, and D. Zagorodnov. NAP: Practical fault-tolerance for itinerant computations. In M. G. Gouda, editor, *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, pages 180–189, Austin, Texas, June 1999. IEEE Computer Society Press.
8. H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Department of Computer Science, Darmstadt, Germany, Mar. 1999. Available at `http://www.informatik.tu-darmstadt.de/BS/Gaertner/publications/TUD-BS-1999-02.ps.gz`. A substantially revised version is available upon request from the authors.
9. V. Roth. Mutual protection of co-operating agents. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 277–287. Springer-Verlag, New York, NY, USA, 1999.
10. K. Rothermel and M. Straßer. A Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents. In *Proc. 17th IEEE Symposium on Reliable Distributed Systems 1998 (SRDS'98)*, pages 100–108, Los Alamitos, California, 1998. IEEE Computer Society Press.
11. T. Sander and C. F. Tschudin. Protecting mobile agents against malicious hosts. In Vigna [13].
12. S. Schneider. Formal analysis of a non-repudiation protocol. In *PCSFW: Proceedings of The 11th Computer Security Foundations Workshop*, pages 54–65. IEEE Computer Society Press, 1998.
13. G. Vigna, editor. *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
14. H. Vogt, H. Pagnia, and F. C. Gärtner. Modular fair exchange protocols for electronic commerce. In *Proceedings of the 15th Annual Computer Security Applications Conference*, pages 3–11, Phoenix, Arizona, Dec. 1999. IEEE Computer Society Press.
15. U. G. Wilhelm. *A Technical Approach to Privacy based on Mobile Agents protected by Tamper-resistant Hardware*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, May 1999.
16. U. G. Wilhelm, L. Buttyàn, and S. Staamann. On the problem of trust in mobile agent systems. In *Symposium on Network and Distributed System Security*, pages 114–124. Internet Society, Mar. 1998.