On the Impossibility of Fair Exchange without a Trusted Third Party

Henning Pagnia
and
Felix C. Gärtner

Darmstadt University of Technology

Department of Computer Science
Technical Report TUD-BS-1999-02

March 18, 1999

Abstract

We attempt to formally define the *strong fair exchange* problem and present a proof that it is impossible to solve strong fair exchange without a trusted third party. The proof is established by relating strong fair exchange to the problem of consensus and adapting the impossibility result of Fischer, Lynch and Paterson. We show that strong fair exchange is at least as hard as consensus and explore a few requirements for trusted third parties in order to be of use in fair exchange.

Categories and Subject Descriptors: C.2.2 [**Computer System Organization**]: Computer-Communication Networks—*Network Protocols*; C.4 [**Computer System Organization**]: Performance of Systems—*Fault tolerance*; F.3.1 [**Theory of Computation**]: Logics and Meanings of Programs—*Specifying and Verifying and Reasoning about Programs*

General Terms: Algorithms, Design, Verification

Additional Key Words and Phrases: consensus, decision problems, electronic commerce, fair exchange, strong fairness, weak fairness

---

The work of the second author was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the "Graduiertenkolleg ISIA" at Darmstadt University of Technology.

Address: Computer Science Department, Darmstadt University of Technology, D-64283 Darmstadt, Germany, Email: `{pagnia|felix}@informatik.tu-darmstadt.de`

1. INTRODUCTION

There are numerous notions of fairness in distributed systems, most of which are related to the scheduling of concurrent actions [Francez 1986]. Lately, there has been an interesting adaption of the term to protocols used in the field of electronic commerce [Asokan 1998]. There, fairness refers to the equal treatment of parties (or agents) which are involved in some form of electronic business transaction.

The basic operation necessary in the usual form of commercial interaction is the exchange of one item for another. This can be a *purchase* of some item in exchange for payment, or the *signing of a contract*, where signed terms of agreement are exchanged. In his recent thesis, Asokan [1998] defines fairness as a security requirement on exchanges:

> An exchange is fair if at the end of the exchange, either each player receives the item it expects or neither player receives any additional information about the other's item. [Asokan 1998, p. 8]

While there are some weaknesses in this definition (see for example the discussion in Pagnia and Vogt [1999]), this notion of fairness can be used to define a new problem in distributed systems: the problem of fair exchange between two parties.

There exist a variety of fair exchange protocols in the literature, all with their own specifications and system models [Camp et al. 1996; Tygar 1996; Asokan et al. 1997; Asokan et al. 1998a; Asokan et al. 1998b; Pagnia and Vogt 1999; Vogt and Pagnia 1999]. A unifying aspect of these protocols however is that — apart from the two involved parties P and Q that exchange an item — almost all of the protocols refer to an additional reliable entity which supports the completion of the protocol in some cases. This entity is commonly referred to as a *trusted third party*, *trustee* or *judge*. It has been an open question whether or not fair exchange is possible without such a trusted third party. In this paper, we explore this question for one particular system model and one formalization of fair exchange along the lines of Asokan [1998]. We will show, that under certain assumptions fair exchange is in fact impossible by relating the problem to other problems in distributed systems. As this is only a preliminary report of current research results, some questions remain open. We state these questions at the relevant places (marked using marginal notes) and thereby hope to spawn some discussion and further work on this subject. To remain brief in this paper, we also omit an exhaustive discussion of the related literature [Camp et al. 1996; Tygar 1996; Asokan et al. 1997; Asokan et al. 1998a; Asokan 1998; Asokan et al. 1998b; Pagnia and Vogt 1999; Vogt and Pagnia 1999]. We plan to return to these tasks soon.

open question

This report is structured as follows: first we present the system model used in this paper in Section 2. A formalization of strong fair exchange in this system model and the impossibility result is then presented in Section 3. Section 4 contains some remarks as to what qualities a trusted third party must have in order to effectively support strong fair exchange. Finally, Section 5 concludes this report.

2. SYSTEM MODEL

In this section, we try to formalize the system model presented in Asokan [1998].

A *system* is a "thing" that interacts with its environment in a discrete fashion across a well-defined boundary (called *interface*) [Lamport 1989]. Specifications for such a system identify the intended behavior at the interface. We will take the state-based approach to specifying such a system [Lamport 1989] and so we have the notion of a *state* which completely describes the state of the interface at any given point in time. The state consists of *input variables* and *output variables*. Both types of variables may be read by both environment and system, but input variables may only be written by the environment whereas output variables may only be written by the system. If this is done, the interface state changes and we say that a *state transition occurs*. A *behavior* of the system is a sequence of states obtained from observing a sequence of state transitions. A *property* is defined as a set of behaviors. A system by itself defines a property, which is the set of all possible behaviors which may be observed at its interface. A property p is said to *hold* for some system, if the set of behaviors defined by the system is a subset of p. Since we are only concerned with *problems*, we do not need to define *programs* or *protocols* which run within the system and produce behaviors at the interface. In our sense, a *specification* S is a property describing those behaviors that are intended at the interface. A program p is said to be *correct regarding* S iff S holds for p. A specification usually consists of two properties: an environment assumption E and a system guarantee M. This signifies that all behaviors are correct iff they are in M or not in E. This underlines the contract nature of a specification: if a specification S holds for some system, then we know that the system will guarantee M if the environment guarantees E.

There are many ways to specify the system in the context of fair exchange between two parties. First and foremost, there must be the two involved parties P and Q that want to exchange items by writing and reading some local variables. The entities P and Q can be any autonomous system, e.g., a process, a mobile agent or even a human user sitting in front of a computer. These two agents together with some communication system form the components.

There is a fundamental distinction between modeling the system as an *open* or *closed* system. A system is closed if it does not interact with its environment [Lamport 1989]. In a closed system, both P and Q are fully part of the system, whereas in an open system, P and Q are part of the environment. These two approaches are depicted in Figure 1.

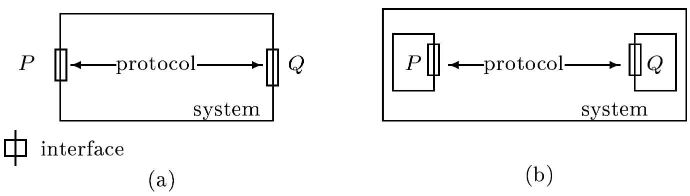

Fig. 1. System for fair exchange scenario: open (a) and closed (b).

In the open system, the interfaces of P and Q can be seen as two remote computer terminals. Thus, P and Q have some "representative" within the system that feeds their input into a message passing protocol. In the closed approach, these representatives are indistinguishable from P and Q themselves. Thus, both approaches are similar but still lead to different type of results.

We will focus on the closed system approach since this seems more natural in the context of distributed systems. We will discuss the consequences of choosing the open system model briefly in Section 4. In the closed system model, both P and Q are modeled as two distinct processes that communicate via an interface through a message subsystem. A specification of a closed system relates the behaviors of internal variables existing at the interfaces of submodules.

As in Asokan [1998], we place no restrictions on the timeliness behavior of the system except that the underlying method of information exchange is reliable (this is the *resilient channel assumption* of Asokan [1998, p. 14]). We further assume that all information passing through the communication system is and will remain confidential (this is the *confidential channel assumption* of Asokan [1998, p. 14]). The only way that P and Q can influence the progress of the system is by writing arbitrary values into their input variables or by completely stopping to interact with the system (this equals the possible actions of both participating parties described by Asokan [1998, p. 14] if one or both parties are dishonest or want to abandon the exchange).

We assume that there is no trusted authority of any kind that either P or Q may turn to in the case they believe that something went wrong during the exchange. This means that there is no additional active entity with an additional interface to or within the system that participates in the exchange. For the moment it suffices to think of this fact as "P and Q being alone" in the system. We will return to the question of what actually is a trusted third party later in Section 4.

3. IMPOSSIBILITY RESULT

In this section, we first formally define the strong fair exchange problem, then define a variation of the consensus problem and relate both to one another. Finally, we discuss the applicability of results concerning consensus to strong fair exchange.

3.1 Specification

We follow the specification of Asokan [1998] for fair exchange with the Strong Fairness property and try to adequately formalize those parts of the specification which remain informal in the text. The same specification also appears in Asokan, Shoup, and Waidner [1998a] and a similar one appears in Asokan, Shoup, and Waidner [1998b].

	P	Q
input	i_P, d_Q, Q	i_Q, d_P, P
output	i_Q with $desc(i_Q) = d_Q$	i_P with $desc(i_P) = d_P$
or	"aborted"	"aborted"

Table 1. Input and output values of fair exchange [Asokan 1998].

3.1.1 *Original Informal Specification.* Table 1 gives the values involved in fair exchange taken from Asokan [1998, p. 9]. It is assumed that there is some function *desc* which maps any exchangeable item to some "description" which may then be compared against some expected description value (in our context this may be a list of item features, a price, etc.). This is necessary since both parties must be able to describe their desired item without actually posessing it. Consequently, apart from the exchangeable items i_P and i_Q both parties possess a description d_Q and d_P of what they expect of i_Q and i_P. Naturally, both parties must know the identities or pseudonyms of the other.

When the fair exchange protocol has terminated, there are two possible outcomes. The first is the case for a successful protocol execution, where both P and Q possess the desired item. The other case is an "unsuccessful" protocol execution where the exchange has not taken place and — instead of receiving i_P and i_Q — both parties receive some indication that the exchange has been aborted. We assume that items are *atomic* in the sense that no part of the item is of any use by itself.

According to Asokan [1998, p. 9f], a protocol which solves fair exchange must have certain properties called *Effectiveness*, *Fairness*, *Timeliness* and *Non-repudiability.* We will omit a discussion of Non-repudiability here because it "is not an integral requirement for fair exchange" [Asokan 1998, p. 10]. Also, there are two notions of Fairness of which we will only consider that of *Strong Fairness.* (We will discuss the notion of Weak Fairness briefly in Section 4.) The properties are described as follows:

—Effectiveness: "If P and Q both behave correctly and do not want to abandon the exchange, then when the protocol has completed, P has i_Q such that $desc(i_Q) = d_Q$ and Q has i_P such that $desc(i_P) = d_P$."

—Strong Fairness: "When the protocol has completed, either P has i_Q such that $desc(i_Q) = d_Q$, or Q has gained no additional information about i_P. The same conditions similarly count for Q."

—Timeliness: "P can be sure that the protocol will be completed at a certain point in time. At completion, the state of the exchange as of this point is either final or changes to the state will not degrade the level of fairness reached so far."

3.1.2 *Formalization.* We now formalize these three properties in the context of the system model described in Section 2.

Since we are in a state-based system, we must re-formulate the above conditions in terms of input and output variables at the protocol interface. So we must think of an exchangeable item and its potential description as some bitstring (i.e., some "number") which is written and stored in some variable.

Table 2 shows the relevant data structures in terms of Asokan [1998]. Here, i_P and i_Q designate input variables of the system in which P and Q write some value (possibly the exchangeable item). Equally, d_Q, d_P, P and Q hold the values described above. There is an additional item store called e_P and e_Q which is an output variable of the system. We assume that e_P and e_Q are in some "empty" state initially and that a subsequent change indicates protocol completion. Upon successful completion, e_P and e_Q will contain the exchanged item. Otherwise they will contain a special marker value indicated as "aborted". We assume that e_P and e_Q are only written once and are never changed thereafter. This assumption is not undue since multiple rounds of the protocol could use version numbers to distinguish the messages of every round. Thus, all messages of a particular round can be held back or discarded if the two parties have agreed on a different round number and started the protocol again. However, we assume that an item is *not revocable* meaning that, for example, once P has aquired knowledge about i_Q, Q can do nothing to destroy the usefulness of i_Q to P.

Description	at P	at Q
exchangable item store (input)	i_P	i_Q
description of wanted item store (input)	d_Q	d_P
partner store (input)	Q	P
exchanged item store (output)	e_P	e_Q
indication of malevolence	m_P	m_Q

Table 2. Data structures of involved processes.

Table 2 also contains an additional entry m_P and m_Q which are boolean values indicating whether P or Q are dishonest or want to abandon the exchange. These values are analogous to the introduction of "error variables" when modelling faults in fault-tolerant computing [Gärtner 1998]. They are needed for formal specification purposes only. If the variable m of a particular process equals false and remains unchanged then we say that the process *behaves non-malevolently* (or simply: *it behaves*). If m equals true at some point during the system execution, then we say that the process *behaves malevolently* (or simply: *it misbehaves*).

We consider each of the above three properties (Effectiveness, Timeliness and Strong Fairness) in turn and derive a formal equivalent which best meets our understanding of the informal descriptions of Asokan [1998].

3.1.2.1 *Effectiveness.* The Effectiveness property is a safety property which ensures that an effective exchange can actually take place. This means that starting with two "well-behaved" nodes P and Q and two "well-shaped" items i_P and i_Q will result in a "well-executed" exchange. Also, the effectiveness requirement ensures that a mismatch of descriptions must end in an abort of the exchange.

—Effectiveness: Both parties receive the other's item if they both behave non-malevolently and the items are the ones which are anticipated. If the items are not the ones expected, then both partners will abandon the exchange. Formally:

$$\big(d_P = desc(i_P) \wedge d_Q = desc(i_Q) \wedge \text{stable}(m_P = m_Q = \mathsf{false}) \Rightarrow e_P = i_Q \wedge e_Q = i_P\big) \wedge$$
$$\big(d_P \neq desc(i_P) \vee d_Q \neq desc(i_Q) \Rightarrow e_P = \text{“aborted”} \wedge e_Q = \text{“aborted”}\big)$$

The predicate stable(ϕ) means that the value of ϕ does not change. We need it to specify the situations in which both partners well-behave. Note that this property does not prescribe what happens if the descriptions match and one or the other party misbehaves. Any outcome of the protocol is allowed as long it complies with the other two requirements.

3.1.2.2 *Timeliness.* The Timeliness property above is a composition of a safety and a liveness property. The liveness part states that the protocol will eventually terminate. The safety part states that upon completion nothing "bad" happens to the state which either partners are in. We have captured this safety part in the assumption that e_P and e_Q are only written once, so the Timeliness requirement becomes a real liveness property.

—Timeliness: Eventually P will write a value to e_P and Q will write a value to e_Q.

The formal version of this property requires the use of some form of temporal logic (such as the $\Diamond$ operator of Pnueli [1977] or the **leads-to** operator of Chandy and Misra [1988]; we omit a treatment of these formalisms since they are not needed for the further understanding of this paper).

3.1.2.3 *Strong Fairness.* While Effectiveness ensures that an effective protocol completion is possible, Strong Fairness rules out some unwanted behaviors where one party gains an advantage over the other.

—(Strong) Fairness: There are no behaviors in which Q receives i_P and P receives "aborted" or P receives i_Q and Q receives "aborted". Formally:

$$\neg((e_P = i_Q \wedge e_Q = \text{“aborted”}) \vee (e_Q = i_P \wedge e_P = \text{“aborted”}))$$

Overall, Effectiveness determines the outcome if descriptions don't match or both descriptions match and both parties are willing to perform the exchange. Fairness globally restricts the outcome to those behaviors that place no disadvantage on either party. Timeliness guarantees eventual termination of the protocol. As usual, it remains an open question whether or not the formal versions of the above properties accurately reflect their meaning, especially the meaning which Asokan [1998] wanted to communicate.

open question

3.2 Decision Problems and Consensus

In this section, we define the class of *decision problems* in the system model presented above. Decision problems are defined for a set of communicating processes in distributed systems [Tel 1994, p. 387] where it is required that each correct (i.e., non-malevolent or non-faulty) process irreversably writes some "decision value" to its output. This decision value usually depends in some form on the input values of the processes. Prominent instances of decision problems are *election* (where it is required that one process decides to become leader and all other correct processes decide to become non-leader) and *consensus* (where all correct processes must decide on a single value).

Consider a distributed system with a set of n processes $p_1, \ldots, p_n$. As we are in a state-based environment, we will define decision problems using two variables π_i and δ_i. Here, π_i is the *propose variable* at process p_i and δ_i is the *decision variable* at p_i. Both can take any value v from a finite set V of decision values. If some value v is placed into π_i, we say that p_i proposes v. Dually, if p_i places some value v in δ_i then we say that p_i decides v. We assume that processes propose and decide at most once and that they must propose a value before they can decide. We also need to define the notion of a *decision vector*, which is a vector $(x_1, \ldots, x_n)$ consisting of the decision values of each p_i (if p_i has not yet decided or is not correct, x_i has some undefined value $\perp$).

An algorithm which solves the decision problem must fulfill the following three correctness conditions:

—Termination: eventually every correct process decides.

—Consistency: at any point in time, the decision vector is an element of some fixed set D of decision vectors.

—Non-triviality: at least two essentially different outcomes of the algorithm must be possible in different executions.

In contrast to Termination (which is a liveness property), Consistency and Non-triviality are both safety properties; while Consistency makes it possible to place restrictions on the decision values of the processes, the Non-triviality condition rules out trivial algorithms where processes do not communicate.

We have already noted that *consensus* is an instantiation of a decision problem: this means that any algorithm that solves the decision problem can also solve consensus. But equally this implies that conditions under which consensus is impossible render solutions to the decision problem impossible.

3.3 Relating Decision Problems to Strong Fair Exchange

There are some obvious analogies between decision problems and strong fair exchange. This becomes even more obvious if we restrict our attention to decision problems for $n = 2$, i.e., with only two processes p_1 and p_2 which play the role of P and Q in fair exchange. As we have specified decision problems using only two variables at one process, we can view the communication subsystem between p_1 and p_2 as the "subsystem" of Figure 1 (b). The "interface" of p_i is an input variable π_i and an output variable δ_i. Obviously they resemble the original item store i_P/i_Q and the exchanged item store e_P/e_Q, respectively.

The obvious analogy in requirements is the liveness property: Timeliness is equivalent to Termination (except for the small word "correct" which appears in Termination and does not appear in Timeliness; we will return to this point shortly). The question remains, how do Consistency and Non-triviality relate to Effectiveness and Strong Fairness?

First of all, the output vectors of decision problems are easily defined for fair exchange. There are only two possible outcomes for (e_P, e_Q), namely (i_Q, i_P) or $(\text{aborted}, \text{aborted})$. Both outcomes are possible as stated jointly by Effectiveness and Strong Fairness, so both imply the Non-triviality condition. But because Effectiveness and Strong Fairness also *completely* define the set of output vectors they also play the role of the Consistency requirement. So at first glance it seems as if strong fair exchange is a decision problem.

There is a slight distinction, however, between decision problems and fair exchange. Decision problems are only interesting to solve if they are studied in faulty environments. The effects of faults onto the behavior of a process are usually defined by a failure model. A simple and intensely studied failure model for example is *crash* in which processes simply stop taking steps forever [Hadzilacos and Toueg 1994]. Depending on the failure model, the specification of decision problems cannot place restrictions on the values decided by faulty nodes. This is why the specification of decision problems always refers to the set of *correct* processes, which — in the case of the *crash* failure model — are all processes that eventually survive.

In fair exchange, we have a "failure model" similar to that of *crash* in that one way to misbehave is simply to stop interacting with the system at all (cf. Section 2). However, the formal fair exchange specification above places a strong restriction on the decision value of a "crashed" (i.e., misbehaving) process, namely, that it must eventually be written. We do not know how the specification of Asokan [1998] is to be understood in this respect? When does a misbehaving peer "know" that the protocol has completed? If strong fair exchange restricts the decision values of misbehaving peers, then it restricts the set of possible decision vectors substantially. But nevertheless, strong fair exchange as specified above is a decision problem.

open question

The failure semantics of a peer in Asokan [1998] can be understood as a process crash or as a process crash with subsequent recovery. The latter is known as the *crash-recovery* failure model [Aguilera et al. 1998]. Since crash-recovery is a more malign failure model than crash (in the sense described in Hadzilacos and Toueg [1994] and Gärtner [1998]) we will assume that the failure model which participating parties follow is in fact crash. So, to make the specification more realistic, we should alter the Timeliness requirement in the following way:

—Timeliness: Eventually a process that behaves non-malevolently will write a value to its exchanged item store e.

This interprets the definition of Asokan [1998] in the following way: "a misbehaving peer does not need to know whether or not the protocol has terminated."

3.4 Fair Exchange as hard as Consensus

In the previous section, we have argued that strong fair exchange is an instance of a decision problem. How do strong fair exchange and consensus relate to oneanother? We show here that strong fair exchange is at least as difficult to solve as consensus. We do this by constructing a consensus algorithm using an algorithm for strong fair exchange. We conjecture that it is impossible to perform the opposite, i.e., build an algorithm for strong fair exchange from a consensus algorithm.

open question

```
function consensus(π ∈ {0,1}) returns δ ∈ {0,1}
  local variable t ∈ {0, 1, "aborted"}
begin
  t := strong_fair_exchange(π, desc(1)); {* 1 *}
  if t ≠ "aborted" then return 1 end
  t := strong_fair_exchange(π, desc(0)); {* 2 *}
  if t ≠ "aborted" then return 0 end
  if π = 1 then
    t := strong_fair_exchange(1, desc(0)); {* 3 *}
    if t ≠ "aborted" then return 0;
      else return π end
  else {* π = 0 *}
    t := strong_fair_exchange(0, desc(1)); {* 4 *}
    if t ≠ "aborted" then return 0;
      else return π end
  end
end
```

Fig. 2. Consensus algorithm using strong fair exchange.

We will construct a consensus algorithm merely involving two parties p_1 and p_2 which want to reach consensus on a boolean value 0 or 1. We assume that both parties have access to a function called *strong_fair_exchange* which takes some exchangeable item and a description of some desired item and returns either the desired item or the special token "aborted". The function is supposed to satisfy the formal specification of strong fair exchange from Section 3.1.

Figure 2 shows the pseudocode of a consensus algorithm built using strong fair exchange. The consensus algorithm takes the proposed boolean value and equally returns the decided boolean value based on the results of the fair exchange function. The algorithm (which was not designed for efficiency) can be described as "gradually increasing knowledge" about the peer process. We assume that at least one process (name it P) behaves non-malevolently. Then there can be exactly six different initial settings which we have to consider. These are shown (together with the return value δ) in Table 3 where "m" signified that Q misbehaves during every invocation of the exchange function.

P	Q	δ
1	1	1
0	0	0
0	1	0
1	0	0
0	m	0
1	m	1

Table 3. Possible initial settings for consensus via fair exchange ("m" signified that Q misbehaves during the protocol execution).

The first invocation of *strong_fair_exchange* will succeed if both parties behave and have proposed "1". This is guaranteed by the Effectiveness requirement and shown in the first line of Table 3. An aborted exchange can only be due to the peer behaving malevolently or either party proposing "0".

The second invocation clarifies this question a little further: if the second exchange succeeds, both parties have proposed "0" (see the second line in Table 3). An aborted exchange can of course mean that the peer has misbehaved. But it can also mean that locally π equals 1 and the peer has proposed "0", or $\pi = 0$ and the peer has proposed "1".

Now we determine whether or not the peer is misbehaving. This is done by checking for the remaining two cases (lines 3 and 4 of Table 3). If the final exchange aborts, then the peer must be misbehaving.

Now we must show that the algorithm in Figure 2 in fact solves consensus, given that *strong_fair_exchange* solves strong fair exchange. Note that we assume that if a peer misbehaves in some invocation of strong fair exchange, then it must also misbehave in all future invocations within the *consensus* function! Obviously, the algorithm is no consensus algorithm if, for example, a peer misbehaves within the first invocation but later consistently proposes $\pi = 1$. **open question**

3.4.1 *Termination.* The Termination condition of consensus states that every correct process eventually decides. Given the Timeliness property of strong fair exchange, it is easy to see that the consensus function in Figure 2 will terminate and thus lead the invoking process to a decision. With the original Timeliness property of strong fair exchange, we actually imposes a decision on *all* processes. The weaker Timeliness requirement of Section 3.3, however, imposes a decision only on the non-misbehaving nodes. As misbehavior mimics crash, the outcome is the real Termination property of consensus.

3.4.2 *Consistency.* The Consistency requirement of consensus is often called *agreement* [Schiper 1997] and states, that all correct processes must decide the same. There are only two parties involved here. If both parties propose "1", then the Effectiveness property of strong fair exchange ensures that the first invocation of *strong_fair_exchange* will succeed at both nodes. Thus, both will decide "1".

If both parties propose "0", then Effectiveness guarantees that the second invocation will succeed at both nodes. Thus both will decide "0".

If the partners propose different values, the next (case 3 or case 4) invocation of the exchange will succeed. In both cases, both parties will decide "0".

Now we only have to consider executions of the protocol where at most one party, say Q, misbehaves. As mentioned above, we assume that a misbehavior within one invocation results in misbehavior within the following invocations of *strong_fair_exchange* as well. Because P is the only process that decides, it can decide arbitrarily. (In the protocol it decides on its own proposed value.)

3.4.3 *Non-triviality.* The Non-triviality condition of consensus is sometimes called *validity* and states that the decided value of any process must have been proposed by some process. Table 3 shows all the possible outcomes of the *consensus* function according to the different initial settings. It shows that the descision value always has been proposed by at least one process. Thus, Non-triviality holds.

Overall, this shows that the algorithm in Figure 2 is a correct consensus algorithm given the correctness of strong fair exchange and the "failure" assumption about the processes. Consequently, strong fair exchange is at least as hard to solve as consensus which is articulated by the following Lemma.

LEMMA 1. *Let S be a system which solves strong fair exchange in the given system model. Then there exists a system C which solves consensus in the same system model.*

3.5 Impossibility

Now we turn to the central result of this report: the impossibility of strong fair exchange in the given system model. We base the result on the widely cited theorem by Fischer, Lynch, and Paterson [1985] (here we follow the presentation given by Tel [1994, p. 393f]).

THEOREM 1. (Impossibility of Distributed Consensus with One Faulty Process [Fischer et al. 1985]) *There is no asynchronous, deterministic, 1-crash-tolerant consensus protocol.*

As we have already shown, strong fair exchange is at least as hard as consensus, i.e., if consensus is impossible to solve in the given system model, then so is strong fair exchange. It remains to validate that we use the same system model as Fischer, Lynch, and Paterson [1985], which we will do now.

In the previous sections, we have made no assumptions about the program nature of the underlying system. Especially, we have implicitly assumed deterministic systems as opposed to probabilistic systems. As Theorem 1 only rules out probabilistic solutions (not nondeterministic ones), we are conform with the system model of Fischer, Lynch, and Paterson [1985] in this respect.

Asokan describes the system model used in his/her thesis as the asynchronous communication model: "we make no timing assumptions such as bounds on message delays, or deviations between local clocks." [Asokan 1998, p. 14] This is exactly the model which Fischer, Lynch, and Paterson [1985] use and which is commonly referred to as the *asynchronous* system model [Schneider 1993].

The term "1-crash-tolerant" refers to the *crash* failure model for processes which has been mentioned earlier in Section 3.3. This means that at any point in time at most one process can simply stop executing steps. The problem with this failure model in the asynchronous case is that a crashed process is indistinguishable from one which is merely "very slow" [Chandy and Misra 1986]. This is the central argument on which the impossibility result of Fischer, Lynch, and Paterson [1985] is based: let p be a process which — by chance — is the first to write its decision value. Because of the non-triviality property, p must wait for at least one message from one of its peers. How long should p wait for this message? If its peer has crashed, p must at some point assume this fact and decide. But if the assumption was wrong (i.e., the peer was only very slow), the decision value of the peer (sent in the message) might be different from the decision value of p, violating agreement. There is no way out of this dilemma!

We have also already noted above that assumptions about the participating parties are similar to the crash failure model for processes. Either P or Q can simply stop participating in the current and any future exchange. We have discussed above that this in fact resembles the process crash failure model.

Overall, we find that the system model is the same, so Theorem 1 gives us:

THEOREM 2. *There exists no asynchronous strong fair exchange protocol tolerant against misbehaving nodes.*

PROOF. Assume that there is a protocol S which solves strong fair exchange. Then due to Lemma 1 there exists a protocol which solves consensus. Because this is a contradiction to Theorem 1, there cannot exist such a protocol S. □

Of course, this result only holds for the system model and the malevolent behavior of exchange participants described above. As in consensus, weakening any of the requirements will probably result in a solvability of the problem. For example, if we model the system as an open system (as indicated in Figure 1 (a)) and we assume that P and Q have no control over the internals of the system, then apart from writing the input variables, P and Q do not further participate actively in the exchange. The system itself is assumed to be reliable. Hence, in contrast to the system model of Fischer, Lynch, and Paterson [1985] the open model may "force" a participant to decide by simply writing some value to e_Q or e_P. Consequently, once P and Q have placed their items into the respective input variables and have indicated that they are willing to start the exchange, the system will bring the exchange to an end no matter what the participants do. This is a clear difference to the model of Fischer, Lynch, and Paterson [1985]. However, in a sense, the system itself (or a part of it) could be viewed as a trusted third party because all that P and Q can see are the values presented at their interfaces (which they must naturally believe). In fact, it seems to us that any requirement which makes strong fair exchange solvable seems to relate to some notion of a trusted third party. Thus, it should be rewarding to study this question further, which we attempt in the following section.

4. WHAT IS A TRUSTED THIRD PARTY?

As discussed at the end of the previous section, modeling the system as a reliable open system makes solutions to strong fair exchange possible if the system itself has some additional properties. If, for example, it may store arbitrary items then the system together with the "representatives" of P and Q behind the interface is exactly the trusted third party of Bürk and Pfitzmann [1990]. Thus we have to exclude such modeling or restrict the system to be a simple message passing subsystem (whatever that means?!). A possible necessary aspect of a trusted third party therefore could be that third parties are able to impose some behavior onto P and/or Q (or their representatives within the system) without their own will. This aspect can also lead to a clarification of another terminological problem: often, a public court, the legal system of a country or some public instutution are referred to as "trusted third parties". These participants clearly lie outside of the system and can only enforce some "state" onto a peer outside of the system.

open question

Often protocols require a trusted third party outside of the system to work. For example, the Weak Fairness requirement of fair exchange is partly based on the fact that a participant "can prove to an arbiter" [Asokan 1998] that the other party has received its item. Still, how to formalize these notions involving *outside* third parties remains an open question to us.

open question

Because of the close relationship between strong fair exchange and consensus it may be interesting to study methods which help solve consensus under the crash failure model. Such methods must make it possible to distinguish a crashed process from a very slow one. The most basic way is to introduce some explicit notion of time into the system [Dolev et al. 1987; Dwork et al. 1988]. So can a reliable source of time be viewed as a trusted third party? Since strong fair exchange is probably harder to solve than consensus, we conjecture that time alone is not enough. But some form of synchrony seems to be necessary.

open question

A different approach to solve consensus is to introduce an implicit notion of time through a construct called a *failure detector* [Chandra and Toueg 1996]. In the context of fair exchange, this would mean to introduce some program component which serves as a "malevolence detector", i.e., it indicates whether the other node is cheating or not. This abstraction seems to fit better into the conceptual framework of electronic commerce, but seems to be of no help in solving strong fair exchange. It is not the question whether or not the peer is cheating or not, which seems important, but rather how to guarantee progress when every participant may fear that its next step will be a loosing one.

open question

Overall, introducing the notion of a trusted third party into a system seems to be equivalent to introducing some explicit or implicit form of synchrony. This results in the following corollary:

COROLLARY 1. *There is no strong fair exchange protocol tolerant against misbehaving nodes without a trusted third party.*

PROOF. A trusted third party is equivalent to introducing some form of synchrony. Thus, if there is no trusted third party we have an asynchronous system. Now Theorem 2 proves the Corollary. □

Asokan [1998] cites a suite of protocols that achieve fair exchange without a trusted third party, which are called *gradual exchange protocols*. This seems to contradict our findings. However, the idea of gradual exchange protocols is to repeatedly exchange "small" parts of a larger item so that a possible unfair move by a peer does not cause "too much damage". This means that the advantage that a peer can achieve over the other can be made arbitrary small. However, there is some point in the protocol after which a peer receives "the final bit" of information that makes the other's item useful. By stopping the protocol at this point, the misbehaving peer can achieve full advantage over the other. Thus, the final bit should be exchanged fairly. Hence, gradual exchange can be reduced to strong fair exchange and consequently does not contradict our results.

5. CONCLUSIONS

We have formally defined the problem of strong fair exchange in distributed systems and have argued that it is impossible to solve without a notion of a trusted third party. We are not aware of any other formal definition of the problem and so we are at risk that our understanding of the subject matter does not match the intuition of other authors in the field. Nevertheless, we think that a formal definition of the problem can contribute to a deeper understanding of the (im)possibilities of exchange protocols in electronic commerce.

As this is only a preliminary research report, the exposition has been unpolished, we have come across a lot of open questions, many ideas remain sketchy and deserve some additional attention. We plan to consider these questions in more detail in the future.

REFERENCES

AGUILERA, M. K., CHEN, W., AND TOUEG, S. 1998. Failure detection and consensus in the crash-recovery model. In *Proceesings of the 12th International Symposium on Distributed Computing (DISC)* (Sept. 1998), pp. 231–245.

ASOKAN, N. 1998. *Fairness in electronic commerce*. Ph. D. thesis, University of Waterloo.

ASOKAN, N., SCHUNTER, M., AND WAIDNER, M. 1997. Optimistic protocols for fair exchange. In T. MATSUMOTO Ed., *4th ACM Conference on Computer and Communications Security* (Zurich, Switzerland, April 1997), pp. 8–17. ACM Press.

ASOKAN, N., SHOUP, V., AND WAIDNER, M. 1998a. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy* (May 1998), pp. 86–99. Printed version contains some errors. Errata sheet is distributed together with the electronic version.

ASOKAN, N., SHOUP, V., AND WAIDNER, M. 1998b. Optimistic fair exchange of digital signatures. In K. NYBERG Ed., *EuroCrypt 98*, Lecture Notes in Computer Science (1998), pp. 591–606. Springer-Verlag. A longer version is available as Technical Report RZ 2973 (#93019), IBM Research, November 1997 at http://www.zurich.ibm.com/Technology/Security/publications/1997/ASW97b.ps.gz.

BÜRK, H. AND PFITZMANN, A. 1990. Value exchange systems enabling security and unobservability. *Computers & Security 9*, 8, 715–721.

Camp, J., Harkavy, M., Tygar, J. D., and Yee, B. 1996. Anonymous atomic transactions. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce* (Nov. 1996), pp. 123–133.

Chandra, T. D. and Toueg, S. 1996. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM 43*, 2 (March), 225–267.

Chandy, K. M. and Misra, J. 1986. How processes learn. *Distributed Computing 1*, 40–52.

Chandy, K. M. and Misra, J. 1988. *Parallel Program Design : a Foundation.* Addison-Wesley, Reading, Mass.

Dolev, D., Dwork, C., and Stockmeyer, L. 1987. On the minimal synchronism needed for distributed consensus. *Journal of the ACM 34*, 1 (Jan.), 77–97.

Dwork, C., Lynch, N., and Stockmeyer, L. 1988. Consensus in the presence of partial synchrony. *Journal of the ACM 35*, 2 (April), 288–323.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM 32*, 2 (April), 374–382.

Francez, N. 1986. *Fairness.* Texts and Monographs in Computer Science. Springer-Verlag.

Gärtner, F. C. 1998. Specifications for fault tolerance: A comedy of failures. Technical Report TUD-BS-1998-03 (Oct.), Darmstadt University of Technology, Darmstadt, Germany.

Hadzilacos, V. and Toueg, S. 1994. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425 (May), Cornell University, Computer Science Department.

Lamport, L. 1989. A simple approach to specifying concurrent systems. *Communications of the ACM 32*, 1 (Jan.), 32–45.

Pagnia, H. and Vogt, H. 1999. Exchanging goods and payment in electronic business transactions. In *Proceedings of the third European Research Seminar on Advances in Distributed Systems (ERSADS)* (Madeira Island, Portugal, April 1999). to appear.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)* (Providence, Rhode Island, Oct. 31–Nov. 2 1977), pp. 46–57. IEEE: IEEE Computer Society Press.

Schiper, A. 1997. Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing 10*, 3, 149–157.

Schneider, F. B. 1993. What good are models and what models are good? In S. Mullender Ed., *Distributed Systems* (Second ed.)., Chapter 2, pp. 17–26. Addison-Wesley.

Tel, G. 1994. *Introduction to Distributed Algorithms.* Cambridge University Press.

Tygar, J. D. 1996. Atomicity in electronic commerce. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC96)* (New York, May 1996), pp. 8–26. ACM.

Vogt, H. and Pagnia, H. 1999. Fairer Austausch beim elektronischen Einkauf im Internet. In *Proceedings of the 6th DFN-CERT Workshop "Sicherheit in vernetzten Systemen"* (Hamburg, Germany, March 1999). in German.