Reducing Fair Exchange to Atomic Commit

Gildas Avoine[1], Felix Gärtner[4], Rachid Guerraoui[2],
Klaus Kursawe[3], Serge Vaudenay[1], and Marko Vukolic[2]

[1] Security and Cryptography Laboratory, EPFL, Switzerland
[2] Distributed Programming Laboratory, EPFL, Switzerland
[3] Kursawe Consulting, Zurich, Switzerland
[4] Dependable Distributed Systems Laboratory, RWTH Aachen University, Germany
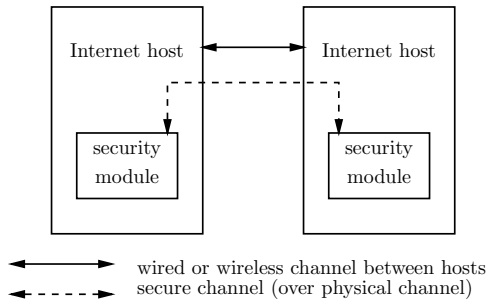
Abstract. The *fair exchange* problem is key to trading electronic items
in systems of mutually untrusted parties. We consider modern variants of
such systems where each party is equipped with a tamper proof security
module. The security modules trust each other but can only communicate
by exchanging messages through their host parties. These are untrusted
and could intercept and drop those messages. We show that the fair
exchange problem at the level of untrusted parties can be reduced to an
atomic commit problem at the level of trusted security modules. This
reduction offers a new perspective with which fair exchange protocols
can be designed. In particular, we present a new atomic commit protocol,
called Monte Carlo NBAC, which helps build a new and practical fair
exchange solution. The exchange does always terminate and no party
commits the exchange with the wrong items. Furthermore, there is an
upper bound on the the probability that the exchange ends up being
unfair, and this bound is out of the control of the untrusted parties.

1 Introduction

1.1 Motivation

An increasing fraction of interactions between processing entities in computer
networks is performed between parties which do not necessarily trust each other.
Mutual distrust however limits the ability to perform useful protocol interac-
tions, e.g., reliably and fairly exchanging valuable electronic items between each
other [16]. Recently, manufacturers have begun to equip hardware with *security
modules* such as a smart card or a special microprocessor. These are assumed
to be tamper proof and run a certified piece of software (see Fig. 1). Examples
include the "Embedded Security Subsystem" within the recent IBM Thinkpad
or the IBM 4758 secure co-processor board [7]. A large body of computer and
device manufacturers has founded the Trusted Computing Group (TCG) [20] to

promote this idea. Because their hardware is tamper proof, their software is certified and they can communicate though secure channels. The major issue at the level of these security modules is the possibility for their untrusted hosts to cut the communication channels and drop their messages. The hosts cannot however alter those messages or pretend to be such security modules. As an interesting consequence, computations between security modules operate in a setting where the only possible faults have benign effects, namely message omissions.

Fig. 1. Hosts and security modules.

Since distributed computing in the presence of benign faults is a well investigated domain, we ask, how solutions to problems between security modules can help solve seminal problems in the security domain (i.e., between untrusted hosts). The problem we consider here is *fair exchange* and we consider it in a synchronous system. Roughly speaking, the problem consists for several untrusted hosts to exchange electronic items. These items should match some specification and either all hosts should terminate the protocol with the expected items or none of them should.

1.2 Contributions

We first present a simple reduction theorem: we show that *fair exchange* at the level of untrusted hosts can be solved if *non-blocking atomic commit* (NBAC) can be solved at the level of security modules. In a sense, we reduce a challenging problem in the area of security [16], to a well known problem in the area of distributed computing [10, 11, 18]. At first glance, the reduction might not look of practical interest because NBAC is a hard problem even at the level of trusted components. In fact, NBAC is impossible if communication channels are unreliable [10]. As described above, the security modules, even if tamper proof, can only communicate through their associated untrusted hosts (see Fig. 1) which can drop the messages and render the communication unreliable. However, we define a new and weaker variant of NBAC that is solvable in this setting and useful in the reduction.

More precisely, we define a probabilistic variant of NBAC, called *Monte Carlo NBAC*. This variant is used in the reduction to build a probabilistic form of fair exchange. Monte Carlo NBAC has a unique combination of properties: it is novel in that it weakens the *agreement* aspect of NBAC (i.e., the fact that all parties must agree on an outcome). More precisely, instead of requiring that no disagreement is possible, we simply require that the probability of disagreement be bound by a constant (that must be out of the control of the adversary). Neither *termination* (i.e., the fact the eventually the algorithm terminates for all parties) nor *validity* (i.e., the fact that the algorithm should abort if some of the items are missing or do not match the required specification) is weakened. The combination we propose also indicates that seeing NBAC as a security building block can still give a new view on this old and seemingly well-understood problem.

We then describe a new atomic commit algorithm that solves Monte Carlo NBAC. This algorithm, together with our reduction result, yields a new fair exchange protocol based on tamper proof security modules. Roughly speaking, the basic idea of the protocol is the following. The security module that initiates the exchange selects, in a random manner, an integer k that it disseminates in a circular manner to all other security modules. The circularity of the dissemination is also based on a random choice of coordinators. The integer k defines the number of rounds through which the security modules need to go successfully (i.e., with no message omission) before committing the protocol. Of course, untrusted hosts can cut the communication at any time and lead to an abort of the exchange, but the probability that they do so in the last round (k), rendering the exchange unfair, is out of their control. In short, our protocol ensures that: (a) all parties are always guaranteed to terminate the exchange; (b) no party ever commits the exchange if the items to be delivered are not those expected; and (c) the probability for the exchange to be unfair has a constant upper bound that cannot be controlled by the *adversary*, i.e., by dishonest parties.

1.3   Related work

It was shown in [8] that fair exchange between two processes is impossible to solve deterministically without a *trusted third party*. *With* a trusted third party, all participants can send their items and descriptions to that party, which checks them and possibly forwards them to the participants [5]. Later, *optimistic* protocols were developed in which participation of the trusted third party is only necessary if something goes wrong [1]. The context of this paper is one where the trusted third party is a virtual entity, somehow distributed overall several security modules. The need to go through untrusted hosts to access these modules makes the use of this virtual trusted third party non-trivial.

In fact, there has been recent work on fair exchange using, instead of a trusted third party entity, different forms of distributed trusted hardware, e.g., smart cards [22] or trusted processing environments in the context of mobile agents [17]. The architecture presented in [6] also provides means to implement synchronous channels between distributed trusted computing compartments. In

[2] a solution to two-party fair exchange is described in a system with tamper proof components using an underlying *synchronization* protocol. None of those approaches explicitly consider reducing fair exchange at the level of untrusted hosts to a clearly defined distributed computing problem (e.g., NBAC) at the level of security modules subject to benign faults.

The notion of *atomicity* was identified to be central to electronic commerce in [21], and the link between fair exchange and atomic commit was observed in [13,14], where an intermediate problem was described (in an intuitive manner) in the context of a distributed model with trusted third parties. We believe however our paper to be the first one to precisely reduce the fair exchange problem into some agreement problem of traditional distributed computing (i.e., NBAC), and make use of solutions to the latter to solve the former.

In [9] it was shown that in a synchronous system with cryptography a majority of honest processes can simulate a centralized trusted third party (and hence solve fair exchange). In [3], algorithms that tolerate benign failures are automatically translated into algorithms that tolerate arbitrary failures (i.e., where parties can be dishonest). The safety of the translation assumes two-thirds of the parties to be honest. Our context is different because we assume every party to include a secure module within it and consider message omissions even between honest participants. As a result, we do not need the honest (two-third) majority assumption.

It is also important to notice that, despite two decades of research on NBAC [10,11,18], including attempts to weaken its validity property to make the problem solvable in practical settings, we know of no work on weakening the agreement part of NBAC and making it probabilistic. Viewing NBAC as a security abstraction provides an ideal context to revisit the problem from this perspective. The new NBAC algorithm we come up with in this paper, namely our Monte Carlo NBAC algorithm, includes some known techniques from deterministic distributed computing (e.g., round-based information exchange), combined with some randomization techniques (e.g., the choice of a random leader or a random number of rounds).

1.4 Roadmap

We present our system and adversary model and relate it to the trusted hardware scenario sketched above in Sect. 2. Sect. 3 recalls fair exchange and NBAC, gives the basic reduction of fair exchange to NBAC, then introduces Monte Carlo NBAC. In Sect. 4 we describe a new fair exchange protocol based on a Monte Carlo NBAC algorithm. We conclude the paper in Sect. 5 with some open questions.

2 Model

2.1 Processes and channels

The system consists of a set of processes interconnected by a communication network with bidirectional channels. Two processes connected by a channel are

said to be adjacent. We assume that the processes are synchronous and the channels are synchronous and secure: between any two adjacent processes P and Q, the following properties are guaranteed:

- (Authenticity) If a message is delivered at Q, then it was previously sent by P.
- (Confidentiality) Message contents remain secret from unauthorized entities.
- (Integrity) Message contents are not tampered with during transmission, i.e., any change during transmission will be detected and the message will be discarded.
- (Synchrony) If a message is sent by P to Q and Q is ready to receive the message, then the message will be delivered at Q within some known bound Δ on the waiting time.

2.2 Untrusted hosts and security modules

The set of processes is divided into two disjoint classes: *untrusted hosts* (or simply *hosts*) and *security modules*. We assume that there exists a fully connected communication topology between the hosts, i.e., any two hosts are adjacent. Furthermore, we assume that every host process P_A is adjacent to exactly one security module process G_A (i.e., there is a bijective mapping between security modules and hosts). In this case we say that P_A is *associated with* G_A (i.e., G_A is P_A's associated security module). No two security modules are adjacent. In other words, for any two security modules G_A and G_B to communicate, they need to do so through their hosts P_A and P_B. This indirection provides the abstraction of an overlay network at the level of security modules. We call the part of the system consisting only of security modules and the virtual communication links between them the *security subsystem* (see Fig. 2). We call the part of the system consisting only of hosts and the communication links between them the *untrusted system*. The notion of association can be extended to systems, meaning that for a given untrusted system, the *associated security subsystem* is the system consisting of all security modules associated to any host in that untrusted system.

2.3 Trust and adversary model

The model sketched above can be related to the setup in practice (given in Fig. 1) as follows: untrusted hosts model Internet hosts and their users, whereas security modules abstract tamper proof components of user systems (like smart cards). Intuitively, security modules can be trusted by other security modules or hosts, and hosts cannot be trusted by anybody. Hosts may be malicious, i.e., they may actively try to fool a protocol by not sending any message, sending wrong messages, or even sending the right messages at the wrong time. We assume however that hosts are computationally bounded, i.e., brute force attacks on secure channels are not possible.Security modules are supposed to be cheap devices without their own source of power. They rely on power supply from their hosts. In principle, a host may inhibit *all* communication between its associated
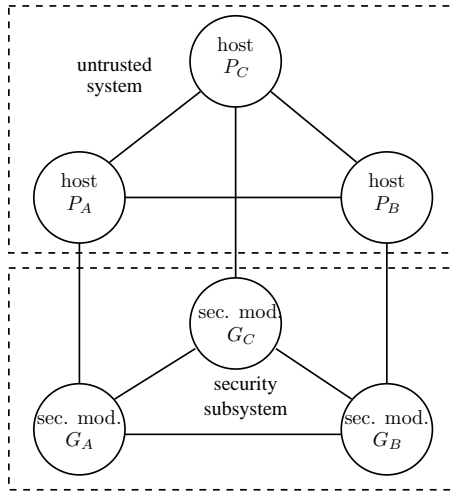
Fig. 2. Hosts and security modules.

security module and the outside world yielding a channel in which messages can be lost. This is a worst-case albeit realistic assumption.[1]

A host *misbehaves* if it does not correctly follow the prescribed protocol. Otherwise it is said to be *honest*. Misbehavior is unrestricted (but computationally bounded as we pointed out). Security modules always follow their protocol. If security modules are studied in systems in which their associated hosts can inhibit all communication, this results in a system model of security modules with unreliable channels (i.e., where messages may not be delivered). In such systems the term *failure* covers such communication failures.

3   Reducing Fair Exchange to Atomic Commit

In the following, we recall the fair exchange and non-blocking atomic commit (NBAC) problems and we show that a solution to NBAC at the level of the security subsystem can be extended into a solution to fair exchange at the level of the untrusted system. In other words, we reduce fair exchange at the level of hosts into NBAC at the level of security modules.

3.1   Fair exchange

Fair exchange is a fundamental problem in any system with electronic business transactions and has attracted a lot of attention recently (see [16] for a comprehensive survey). In fair exchange, we assume that the participating processes

---

[1] In the best case, security modules may possess their own way of independent communication (like in the security kernel approach of the *timely trusted computing base* [6]) yielding a fully reliable synchronous channel, or provide incentives for the host to support timely communication.

start with an item they want to trade for another item. They additionally possess an executable description of the desired item (i.e., a boolean function with which an arbitrary item can be checked for the desired properties). Furthermore, they know from which process to expect the desired item and which process is expecting their own item.

Definition 1 (fair exchange). *A protocol solves* fair exchange *if it satisfies the following properties:*

- *(Timeliness) Every honest process eventually terminates.*
- *(Effectiveness) If no process misbehaves and if all items match their descriptions then upon termination each process has the expected item.*
- *(Fairness) If the desired item of an honest process does not match its description or any honest process does not obtain any (useful) information about the expected item, then no process can obtain any (useful) information about any other process' item.*

The Timeliness property ensures that every honest process can be sure that at some point in time the protocol will terminate. The Effectiveness property states what should happen if all goes well. Finally, the Fairness property postulates restrictions on the information flow for the case where something goes wrong in the protocol.[2] Note that the first precondition of the Fairness property ("if the desired item of an honest process does not match the description...") is very important. Without this condition, a "successful" outcome of the exchange would be possible even if an item does not match the description, which should clearly be considered unfair.

3.2 Non-Blocking Atomic Commit

NBAC is well-known from the distributed computing literature [4, 11, 12, 18]. In NBAC, the set of participating processes start each with a proposed value (usually either *yes* or *no*), and tries to reach a common decision (usually *commit* or *abort*). NBAC is the basic problem to be solved when implementing a distributed database transaction. In the following, we give a definition of the problem in a distributed context where the source of failures is message omission, e.g., processes do not really crash per se, but their messages might no reach their destination.

Definition 2 (NBAC). *A protocol between n processes solves* NBAC *if it satisfies the following properties:*

- *(Termination) Every process eventually reaches a decision.*
- *(Agreement) No two processes decide differently.*

---

[2] We use here the concept of information flow to define fairness in a way that cleanly separates the distinct classes of *safety*, *liveness*, and *security* properties in the specification of the problem [15].

- *(C-Validity) If all processes propose yes and there is no failure then the decision value must be commit.*
- *(A-Validity) If at least one process proposes no then the decision value must be abort.*

3.3 Reduction

We now show that a solution to NBAC at the level of the security subsystem can be extended into a solution to fair exchange at the level of the untrusted system. The derived solutions make no use of an explicit trusted third party within the trusted system. In a sense, the security subsystem acts as a distributed trusted third party.

Theorem 1. *If NBAC is solvable in the security subsystem, then fair exchange is solvable in the associated untrusted system.*

Proof. Given an algorithm to solve NBAC, we construct a protocol to solve fair exchange. The local API of NBAC is a function *NBAC(vote)* which takes a *yes/no* vote and upon termination returns either *abort* or *commit*.

So assume that we have a solution to NBAC in the security subsystem consisting of security modules $G_1, \ldots, G_n$. Now consider the protocol depicted in Fig. 3. This is a wrapper around the solution which can be implemented within the security modules and offers the interface of fair exchange to the hosts. In the protocol, a host hands its item and the executable description of the desired item to the associated security module. The security module exchanges the item with its partners, then checks the item. Finally all security modules agree on the outcome using the underlying NBAC algorithm. The proposal value for NBAC is *yes* iff the check was successful and no abort was requested by the host in the meantime. If NBAC terminates with *commit*, then the security module releases the item to the host.[3] We now discuss each of the properties of fair exchange.

---

[3] Another problem often encountered in the distributed computing literature is *consensus*. (Binary) consensus is similar to NBAC in that it also consists of Termination and Agreement properties. The proposal and decision values are also from the set $\{0, 1\}$. However, the Validity property has no built-in affinity towards *abort* like in NBAC. It just states that if all processes propose the same value then that value must be the decision. So if two processes propose different values then both these values can become decision values. One can ask whether our reduction also works using consensus (e.g., replacing the call to *NBAC* with a call to *consensus* and replacing *yes/commit* with 1 and *no/abort* with 0). This is not possible, as we now explain. Consider an execution in which all hosts are honest, and all items except one match their description. From the protocol, one host will propose 0 and all others will propose 1. The Validity property of consensus does not determine the value of the decision (since both 0 and 1 have been proposed). So assume that consensus terminates with a decision value of 1. This means that the exchange will terminate successfully and the items will be released to the hosts. This violates Fairness, since one item does not match its description and so no process should learn anything about any other item.

Consider the Timeliness property of fair exchange, and observe that the only point in which a honest host may block is while it is waiting for the expected item from its partners (the Termination property of NBAC ensures that it cannot block in the final part of the protocol). However, the receipt of the message is guarded by a timeout, so the process never blocks indefinitely (remember that we assume a synchronous system). Consider Effectiveness and assume that all participating hosts are honest and all items match their descriptions. Because channels are synchronous, processes will not time out prematurely and so every process receives the expected item and successfully validates the description. Hence, all the votes for NBAC will be *yes*. From Agreement and C-Validity, the outcome must therefore be *commit* at all participating processes. Hence, every host will receive the expected item. Consider now Fairness and observe that, in our adversary model, no misbehaving host can derive any useful information that is exchanged over the secure channels. The only way to receive information is through the interface of the *FairExchange* procedure. If one item does not match the description at some honest host, then that host will engage in NBAC with a vote of *no*. A-Validity of NBAC implies that the exchange results in *abort* so that no process receives anything from the exchange. Additionally, if some honest host receives nothing through the exchange, then the Agreement property of NBAC implies that nobody can receive anything. □

FairExchange(*item i, description d*) **returns** *item* {
   ⟨send *i* to exchange partners over secure channel⟩
   timed wait for ⟨expected item i_e from exchange partners over secure channel⟩
   ⟨check *d* on i_e⟩
   if ⟨check succeeds and no timeout⟩
   then *vote* := *yes* **else** *vote* := *no* **endif**
   result := *NBAC(vote)*
   if *result* = *commit* **then return** i_e **else return** ⟨abort⟩ **endif**
}

Fig. 3. Using NBAC to implement fair exchange: code of every host.

3.4   Weakening NBAC

Clearly, our reduction indicates that any solution to NBAC (within the security subsystem, i.e., at the level of the security modules) can lead to a solution to fair exchange (within the untrusted system, i.e., at the level of the hosts). Unfortunately, NBAC cannot be solved deterministically in the secure subsystem because of the possibility of message omissions [10]: remember that hosts can—in the general adversary model—at any time cut the underlying channels connecting security modules. However, we introduce here a probabilistic variant of NBAC, the solution of which leads to a practical solution to fair exchange, as we discuss in the next section.

The variant of NBAC we consider is probabilistic only in the "safety" aspect of the specification. Following the classification of [19] such a specification falls into the class of *Monte Carlo* algorithms.

Definition 3 (Monte Carlo NBAC). *An algorithm solves* Monte Carlo NBAC *if it satisfies the following properties:*

- *Termination of NBAC.*
- *A-Validity of NBAC.*
- *C-Validity of NBAC.*
- *Agreement of NBAC with some probability p ($0 < p < 1$) and p is out of the control of the adversary.*

Recall the adversarial model which we are dealing with: a set of hosts can act maliciously. We assume that these hosts know the NBAC algorithm running in the security subsystem, and may cut channels and hence drop messages at any time. Our requirement that the probability of violating Agreement be out of the control of the adversary means that, even if the adversary knows the algorithm executed by the security modules, the hosts cannot, by dropping messages at specific points of the algorithm execution, render this probability sufficiently low.

Having p out of control of untrusted hosts is important if we use Monte Carlo NBAC in the reduction described in Theorem 1: the Agreement property (in the presence of misbehaving hosts) is only important in the proof of the Fairness property. Revisiting the proof of Theorem 1, it is easy to see that the probability of achieving Agreement directly translates to the probability of Fairness. Hence, if this probability is out of the control of the adversary, the probability that Fairness will hold is also outside the control of the adversary.

4 The Monte Carlo Atomic Commit Algorithm

We now propose an algorithm which solves the Monte Carlo NBAC problem between security modules and indirectly helps build a practical fair exchange protocol at the level of the untrusted hosts.

We assume that all processes involved in the algorithm know each other. For simplicity, we denote the involved processes (the security modules) by $G_1, \ldots, G_n$. The process with the lowest number is the initiator. The pseudocode of the algorithm is given in Fig. 4. Roughly speaking, the processes go through a series of round: the number of these rounds and the coordinator of every round are chosen randomly. We describe the algorithm more precisely in the following.

All processes are initialized by default to an *abort* state (*finished = false*). To start the algorithm, the initiator G_1 chooses a random number $k > 1$ as well as a random number i in $\{1, 2, 3, \ldots, n\}$, marks a message m with i and sends m, containing k to G_i. Process G_i is the initiator of the next round: it picks a new random number j (the next process which will receive the message) in $\{1, 2, \ldots, n\} \backslash \{i\}$ and adds a mark j in m. Process G_j picks a new random number

k in $\{1, 2, \ldots, n\} \setminus \{i, j\}$, adds a mark k in m and sends m to G_k. The processes continue doing so until one of them, say G_l receives m that is marked by full set $\{1, 2, 3, \ldots, n\}$. At that point G_l starts a new round by decreasing the round number found in m and picking the next security module from $\{1, 2, 3, \ldots, n\}$. The participants conduct k of such rounds. Upon reception of a message from the last round, the processes go into a *commit* state (but the timer is still reset).

Every message receipt is monitored with a timeout. If no message is received in $(2n-1)\Delta$ time units after the beginning of the algorithm or the most recent message receipt, the process decides according to its current state (*abort* or *commit*). The timeout is also renewed in the final round to prevent hosts from colluding. If any process begins the algorithm with a proposal of *no*, it does not participate (does not forward any messages), and decides *abort* in a unilateral manner.

Theorem 2. *The algorithm displayed in Fig. 4 solves Monte Carlo NBAC.*

Proof. It is easy to see that the protocol satisfies Termination because of the timeouts (remember that we assume a synchronous system). It is also straightforward to see that C-Validity is ensured: if there is no failure and all propose *yes*, they all reach the end of the final round and decide *commit*. Consider A-Validity, and assume that some process proposes *no*. From the algorithm, we know that this process does not participate in the algorithm, and so not even a single round will succeed. Hence, all will timeout and abort.

Consider now Agreement. If some message is lost before the final message round (i.e., a host cuts any channel of its security module), all processes will time out and abort. The only way for Agreement to be violated is for a message to be lost in the final round. The adversary cannot know k from observing the communication channel because the channels are secure. So if k is chosen in a random manner, there are only two cases to consider. (1) The adversary guesses k a priori. This probability depends on the probability distribution of process G_1 choosing k, giving a bound on the probability that Agreement is violated and this is out of the control of the adversary. (2) The second way through which the adversary might learn k without deciphering the secure channel is by observing a posteriori the encrypted actions of every process G_i. As soon as G_i commits, the adversary knows the most recent message belonged to the k-th round. It can then cut the channel to some other process G_j leading it to abort. However, there is a final timeout of $(2n-1)\Delta$ before comitting, which makes sure the messages have been transmitted before any commit. $\square$

Analysis. The number of messages exchanged is $k \cdot n$ for the case where no failure occurs. Since processes exchange messages in a linear order it is possible to give a bound of $(k \cdot n + (2n-1))\Delta$ on the time it takes for the algorithm to terminate (note that the algorithm terminates only after a final timeout of $(2n-1)\Delta$ after receiving the final message). The bit complexity of the messages is linear in n and logarithmic in k.

We now analyze the probability of a violation of Agreement. Let q_i be the probability that G_1 chooses $k = i$, i.e. $\Pr[k = i]$. Assume that the adversary is

NBAC(*vote*) **returns** *decision* {
  finished := *false*
  set_timer($(2n-1)\Delta$)
  if *vote* = *yes* **then**
    if ⟨G_i is initiator⟩ **then**
      ⟨generate random number $k > 1$⟩
      RoundSet := $\emptyset$
      round := *k*
      send_next_message(*round*, *Roundset*)
    end
  end
  wait for ⟨message receipt or expiry of timer⟩
  upon ⟨receipt of [*round*, *RoundSet*] from previous process⟩ **do**
    if *RoundSet* = ⟨set of all participating processes⟩ **then**
      RoundSet := $\emptyset$
      round := *round* $- 1$
    end
    if (*round* > 0) **then**
      send_next_message(*round*, *Roundset*)
    end
    if (*round* = 0) $\vee$ ((*round* = 1) $\wedge$ (*RoundSet* $\neq \emptyset$)) **then**
      reset_timer
      finished := *true*
    end
  end
  upon ⟨expiry of timer⟩ **do**
    if *finished* **then**
      return(*success*)
    else
      return(*abort*)
    end
  end
}

procedure *send_next_message*(*round*, *RoundSet*) { // call by reference
  nextGA := ⟨some participating process not in *RoundSet*⟩
  RoundSet := *RoundSet* $\cup$ {*nextGA*}
  send [*round*, *RoundSet*] **to** *nextGA*
  reset_timer
}

Fig. 4. Pseudocode of the Monte Carlo NBAC algorithm: code of process G_i.

willing to drop the i-th message. The only way to violate Agreement is to have $i = k$. Hence, the maximum probability that the algorithm ends unfairly over all possible misbehaviors of the adversary is $\max_i q_i$. In [2] various probability distributions are studied and it is shown that choosing k uniformly from an interval $[1, \ldots, N]$ minimizes the probability that Agreement is violated. Hence, by choosing $N = 10$ the probability that Agreement is satisfied is 0.9. This analysis indicates a tradeoff between a large value of the probability of ensuring Agreement and the expected number of rounds of the protocol (which is $N/2$ for a uniform distribution from the interval $[1, \ldots, N]$).

5   Conclusions

In this paper, we showed that solutions to fair exchange at the level of untrusted hosts can be derived from solutions to the non-blocking atomic commit (NBAC) at the level of security modules (tamper proofs components within every host). To make the solutions practical, we weakened the specification of NBAC in a novel way, resulting in Monte Carlo NBAC. Using Monte Carlo NBAC, we derived a new and practical solution for fair exchange. In general, we believe the model considered (hosts with security modules) to be very realistic with modern technology, and the reduction to NBAC to open an interesting research direction bridging the gap between security and distributed computing. Many questions are open.

A natural question to ask is whether we could devise more efficient variants of our Monte Carlo NBAC algorithm? At first glance, one might think for instance of limiting the size of the messages and the length of the timeouts by randomly choosing a total number of messages (instead of a total number of rounds) and have a fixed chain of processes through which the messages would flow (instead of randomly chosen coordinators). Messages exchanged will in this case be a logarithmic function of the random number. The trade-off here is that hosts of processes that are early in the chain have a clear advantage over the others leading to an algorithm that is inherently unfair. (This helps explains our motivation for choosing the total number of rounds and the coordinator of every round, both in a random manner). Coming up with optimality results for the complexity of Monte Carlo algorithms is an interesting question. Another natural question to ask is whether other weakened NBAC variants can be used to derive practical solutions to fair exchange.

References

1. 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.

2. G. Avoine and S. Vaudenay. Fair exchange with guardian angels. In *The 4th International Workshop on Information Security Applications – WISA 2003*, Lecture Notes in Computer Science, Jeju Island, Korea, August 2003. Springer-Verlag.

3. R. A. Bazzi and G. Neiger. Simplifying fault-tolerance: providing the abstraction of crash failures. 48(3):499–554, 2001.

4. P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA, 1987.

5. H. Bürk and A. Pfitzmann. Value exchange systems enabling security and unobservability. *Computers & Security*, 9(8):715–721, 1990.

6. M. Correia, P. Veríssimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In *Proc. of the Fourth European Dependable Computing Conference*, Toulouse, France, Oct. 2002.

7. J. G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S. W. Smith, and S. Weingart. Building the IBM 4758 secure coprocessor. *IEEE Computer*, 34(10):57–66, Oct. 2001.

8. S. Even and Y. Yacobi. Relations amoung public key signature systems. Technical Report 175, Computer Science Department, Technicon, Haifa, Israel, 1980.

9. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proceedings of the 19th ACM Symposium on the Theory of Computing (STOC)*, pages 218–229, 1987.

10. J. Gray. Notes on database operating systems. In *Operating Systems — An Advanced Course*, number 66 in Lecture Notes in Computer Science. Springer-Verlag, 1978.

11. R. Guerraoui. Revisiting the relationship between non-blocking atomic commitment and consensus. In J.-M. Hélary and M. Raynal, editors, *Proceedings of the 9th International Workshop on Distributed Algorithms (WDAG95)*, volume 972 of *Lecture Notes in Computer Science*, pages 87–100, Le Mont-Saint-Michel, France, 13–15 Sept. 1995. Springer-Verlag.

12. R. Guerraoui. Non-blocking atomic commitment in asynchronous systems with failure detectors. *Distributed Computing*, 15(1):17–25, 2002.

13. S. P. Ketchpel and H. Garcia-Molina. Making trust explicit in distributed commerce transactions. In *Proceedings of the 16th IEEE International Conference on Distributed Computing Systems (ICDCS96)*, pages 270–281, Hong Kong, May 1996. IEEE Computer Society Press.

14. S. P. Ketchpel and H. G. Molina. A sound and complete algorithm for distributed commerce transactions. *Distributed Computing*, 12:13–29, 1999.

15. J. McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, Jan. 1996. Special Section—Best Papers of the IEEE Symposium on Security and Privacy 1994.

16. H. Pagnia, H. Vogt, and F. C. Gärtner. Fair exchange. *The Computer Journal*, 46(1), 2003.

17. H. Pagnia, H. Vogt, F. C. Gärtner, and U. G. Wilhelm. Solving fair exchange with mobile agents. In *Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents (ASA/MA2000)*, volume 1882 of *Lecture Notes in Computer Science*, pages 57–72, Zurich, Switzerland, Sept. 2000. Springer-Verlag.

18. D. Skeen. Non-blocking commit protocols. In *Proc. ACM SIGMOD Conf.*, page 133, Ann Arbor, MI, Apr.-May 1981.

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

20. Trusted Computing Group. Trusted computing group homepage. Internet: `https://www.trustedcomputinggroup.org/`, 2003.

21. J. D. Tygar. Atomicity in electronic commerce. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC '96)*, pages 8–26, Philadelphia, PA, May 1996. ACM Press.

22. H. Vogt, F. C. Gärtner, and H. Pagnia. Supporting fair exchange in mobile environments. *ACM/Kluwer Journal on Mobile Networks and Applications (MONET)*, 8(2), Apr. 2003.