A Topological Condition for Solving Fair Exchange in Byzantine Environments

Benoît Garbinato and Ian Rickebusch

Université de Lausanne
CH-1015 Lausanne, Switzerland
{benoit.garbinato, ian.rickebusch}@unil.ch

Abstract. In this paper, we study the solvability of fair exchange in the context of Byzantine failures. In doing so, we first present a generic model with trusted and untrusted processes, and propose a specification of the fair exchange problem that clearly separates safety and liveness, via fine-grained properties. We then show that the solvability of fair exchange depends on a necessary and sufficient topological condition, which we name the *reachable majority* condition. The first part of this result, i.e., the condition is necessary, was shown in a companion paper and is briefly recalled here. The second part, i.e., the condition is sufficient, is the focal point of this paper. The correctness proof of this second part consists in proposing a solution to fair exchange in the aforementioned model.

1 Introduction

Intuitively, a fair exchange is an exchange of items among two or more parties where the only possible outcome is either that all parties obtain their items, or none of them do. In our modern daily lives, the notions of fair exchange and trust are ubiquitous: everyday, without even noticing, we participate in numerous commercial exchanges, which we expect to be fair (and most actually are). Such exchanges range from buying a coffee to spending a significant part of our savings in buying a house. A key enabler to make all these exchanges occur is the notion of *trust*. In the physical world, this trust is supported by the identification and the implicit reputation of tangible exchange partners.

In the digital world, on the contrary, fair exchange is a surprisingly difficult problem. This can be explained by the lack of trust that characterizes the digital realm. In an e-commerce environment, an exchange partner behaving unfairly can vanish without a trace, in contrast with a physical commercial environment where a partner can be approached physically and held accountable for a misbehavior. Yet, fair exchange is a fundamental problem that has constantly been studied over the past decades and that has recently regained interest [1,2,3,4]. This is partly due to the advent of m-business as a natural evolution of e-business, i.e., extending the possibilities of e-business through the use of mobile devices, e.g., cellular phones. When it comes to solving fair exchange in such semi-open environments, i.e., where all parties are not necessarily identified a priori, carefully modeling and analyzing trust relationships between peers is a key issue.

© Springer-Verlag Berlin Heidelberg 2006

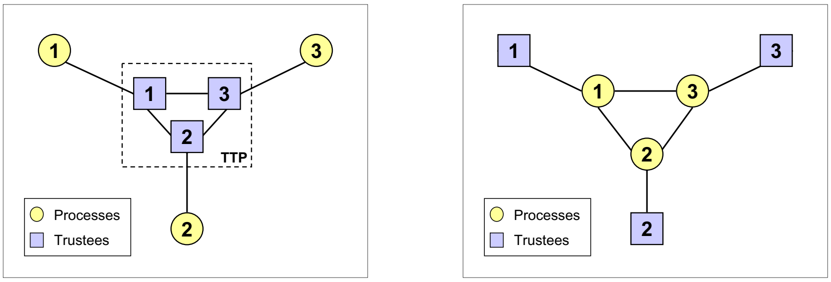
Contribution and Roadmap. This paper propose a generic model in order to study fair exchange in different network settings and also provides a topological condition, both necessary and sufficient, for solving fair exchange. In Section 2, we introduce a synchronous distributed model where processes are divided into two categories, namely *participants*, which can be Byzantine, and *trustees*, which are known a priori to be correct. In that section, we then formally define the fair exchange problem via fine-grained properties that separately capture the liveness and safety requirements of the problem. In Section 3, based on an impossibility result shown in a companion paper, we present a necessary and sufficient topological condition for solving fair exchange in a model with trustees. Section 4 then presents a solution to fair exchange under the aforementioned condition: this solution and its proof provide the correctness proof for the condition. Finally, Section 5 discusses related work, while Section 6 summarizes our contribution and sketches ongoing and future work.

2 Model and Problem Statement

Intuitively, our model consists in a synchronous distributed system composed of two types of processes: *participants*, which are processes potentially subject to Byzantine failures, and *trustees*, which are known a priori to be correct (and which can thus be trusted). The addition of trusted processes in our model is motivated by the fact that fair exchange is impossible in the absence of trust, i.e., without at least one correct process trusted a priori by all other processes [4]. Adding only a single trusted process would however limit the scope of our model and imply a specific role for that trustee, i.e., that of a Trusted Third Party (TTP). For this reason, we associate a trustee with each process, hence uniformly splitting the notion of trust among participants of the exchange and allowing for fully decentralized approaches.

A Generic Yet Realistic Model. The notion of trustees allows us to produce a generic model applicable to various trust and network topologies [2,5]. In particular, this model does not dictate the role of trustees in the fair exchange protocol, i.e., how trustees are connected or the amount of computation they bear. As a consequence, most existing solutions, either centralized or decentralized, can be described in our model. For example, Figure 1(a) shows a classical centralized trust setting, typically via a TTP as in [5], and the equivalent setting in our model. Figure 1(b) then illustrates a distributed trust setting, as with Guardian Angels [2]. By splitting the trust among all participants, via their respective trustees, we can show that the existence of a decentralized solution to fair exchange depends on a rather simple topological condition.

In practice, a trustee is typically implemented via a tamperproof piece of hardware embedded in each host, e.g., a specialized chip or a smart card. This hardware-based approach is gaining momentum in the industry, as illustrated by efforts from IBM, with both its PCI 4758 and PCI-X 4764 cryptographic coprocessors [6], and from Intel, with its Trusted Platform Module [7]. Such solutions are expected to



(a) Classical TTP in our model. (b) Distributed TTP.

Fig. 1. Examples of trust topologies

eventually become mainstream, as the urge to go beyond software-based security increases, in particular in the realm of digital rights management, and as fully decentralized peer-to-peer architectures are being deployed.

2.1 System Model

More formally, we consider a distributed system consisting of a set Π of n *processes*, $\Pi = \{p_1, p_2, \ldots, p_n\}$. Processes of Π are called *participants*. We complete our model with a set Π' of n trusted processes, $\Pi' = \{p'_1, p'_2, \ldots, p'_n\}$, i.e., a *trusted process* is known to be correct a priori by all other processes. Processes of Π' are called *trustees*. Furthermore, each p'_i is matched in a one-to-one relationship with the corresponding participant p_i and is directly connected to it. The set Π^+ is then the set of all $2n$ processes, i.e., $\Pi^+ = \Pi \cup \Pi'$. Participants are processes actually taking part in the exchange by offering and demanding items, and they may exhibit Byzantine behaviors. Trustees on the contrary are *trusted processes* that have no direct interest in the exchange. Their role is to decide when it is appropriate to provide their associated participant with its expected item. We also assume the existence of a *Public Key Infrastructure* (PKI), i.e., each process (participants and trustees) owns a private key and made the corresponding public key accessible to all other processes. Among other things, this assumption provides message unforgeability.

Topology and Synchrony. Processes are interconnected by a communication network and communicate by message passing. The system is *synchronous*: it exhibits *synchronous computation* and *synchronous communication*, i.e, there exists upper bounds on processing and communication delays. To help our reasoning, we also assume the existence of some global real time clock, whose tick range, noted T, is the set of natural numbers.[1]

Regarding the network topology, we assume that processes of Π^+ form a connected graph and that there exists a direct link between any participant and

[1] This global clock is virtual in the sense that processes do not have access to it.

its trustee. Links are reliable bidirectional communication channels, i.e, if both the sender and the receiver are correct, any message inserted in the channel is *eventually* delivered by the receiver. The *synchronous* system assumption further tells us that the delivery will occur within some known time bound Δ_{PL}. Formally, such channels are said to be *perfect links* (PL), which provide *send* and *deliver* primitives (respectively PL.send() and PL.deliver() functions) and ensure the well-known *termination* and *no creation* properties.

Executions and Failure Patterns. We define the *execution* of algorithm A as a sequence of steps executed by processes of Π^+. In each step, a process has the opportunity to atomically perform all three following actions: (1) send a message, (2) receive a message and (3) update its local state.[2] Based on this definition, a *Byzantine process* is one that deviates from A in any sort of way, so a Byzantine process is Byzantine against a specific algorithm A. It is a known result that Byzantine failures can only be defined with respect to some algorithm [8]. A *Byzantine failure pattern* f is then defined as a function of T to 2^{Π} where $f(t)$ denotes a set of Byzantine processes that have deviated from A through time t. In a way, a failure pattern f can be seen as a projection of all process failures during some execution of A. Once a process starts misbehaving, it cannot return to being considered correct, i.e., $f(t) \subseteq f(t+1)$. We also define F as the set of all possible failure patterns of A, so $f \in F$. Let $\mathsf{Byz}(f) = \bigcup_{t \in T} f(t)$ denote the set of Byzantine processes in f. We then define the set F_b of all failure patterns where no more than b processes are Byzantine. More formally, F_b is the largest subset of F such that, for any failure pattern $f \in F_b$, $|\mathsf{Byz}(f)| \leq b$, with $0 \leq b \leq n$:

$$F_b = \{f \in F \ : \ |\mathsf{Byz}(f)| \leq b\} \ \textit{with} \ 0 \leq b \leq n \ .$$

Note that b is bounded by n, the number of processes in Π. From this definition, b is the maximum number of Byzantine processes in any failure pattern of F_b and $F_n = F$. Note that all the above definitions regarding executions and failures are similar to the models of [8,9], but that failures refer exclusively to participants, i.e., processes of Π, since trustees are correct.

2.2 The Fair Exchange Problem

The fair exchange problem consists in a group of processes trying to exchange digital items in a fair manner. The difficulty of the problem resides in achieving fairness. Intuitively, fairness means that, if one process obtains the desired digital item, then all processes involved in the exchange should also obtain their desired digital item. The assumption is made that each process knows both the set Π of processes participating in the fair exchange and the terms of the exchange. The terms of the exchange are defined by a set D of expected item descriptions, $D = \{d_1, d_2, \ldots, d_n\}$, and a set Ω of pairs of processes (p_i, p_j). A description d_i is the description of the item expected by process p_i. Furthermore d_i is unique, so

[2] At each step, the process can of course choose to skip any of these actions, e.g., if it has nothing to send.

if $i \neq j$, then $d_i \neq d_j$. A pair (p_i, p_j) defines the receiver p_j of the item offered by p_i. Elements of Ω are defined such that p_j is the image of p_i through a bijective map (or permutation) of Π, with $i \neq j$. Finally, let M denote the set of digital items m_i actually offered by process p_i during an execution of fair exchange, $M = \{m_1, m_2, \ldots, m_n\}$. Note that, accordingly, for each description in D there does not necessarily exist a corresponding item in M, since M includes items that might have been offered by Byzantine processes. Finally, let $\mathsf{desc}(m)$ be the function returning the description of item m.

Fair Exchange as Service. Fair exchange can be seen as a service allowing processes to exchange digital items in a fair manner. Each process offers an item in exchange for a counterpart of which it has the description. The exchange is completed when every process releases the desired counterpart or all processes release the abort item φ, meaning that the exchange has aborted. To achieve this, the service offers the two following primitives.

offer(m_i, p_j) – Enables the process p_i to initiate its participation in the exchange with processes of Π by offering item m_i to p_j, in exchange for the item matching description d_i, with d_i and Π known a priori.[3]

release(x) – Informs the process that the exchange completed and works as a callback. Process p_i receives item x, which is either an item matching d_i or the abort item φ.

Note that, at the end of an exchange, we say that p_i *releases* an item, meaning that the service calls back the *release* operation of p_i. This convention is similar to the one used for typical *deliver* primitives, e.g., with reliable broadcast primitives [10].

Fair Exchange Properties. We now specify the formal properties of the fair exchange problem. While several other specifications exist in the literature [2,3,11], our specification differs in that it separates safety and liveness via *fine-grained properties.* Such elemental properties then allow us to better reason about the correctness of our solution.

Validity. If a correct process p_i releases an item x, then either $x \in M$ and x matches d_i, or x is the abort item φ.

Uniqueness. No correct process releases more than once.

Non-triviality. If all processes are correct, no process releases the abort item φ.

Termination. Every correct process *eventually* releases an item.

Integrity. No process p_j releases an item m_i, with process p_i correct, if m_i matches description d_k of some correct process p_k, with $p_k \neq p_j$.

Fairness. If any process p_i releases an item m_j matching description d_i, with p_i or p_j correct, then every correct process p_k releases an item matching description d_k.

[3] When defining the FE problem, trustees are not required since they have no direct interest in the exchange.

Among these six properties, the last two, *integrity* and *fairness*, are specific to the problem of fair exchange and define precisely the possible outcomes of fair exchange algorithms. Other specifications of fair exchange usually rely on a single property to capture the notion of fairness [2,5,11]. However we argue that if those specifications are suitable for cases where $n = 2$, they are impossible to satisfy in models allowing more than one Byzantine process. In [2], for example, the *fairness* property requires that if any correct process does not obtain its item, then no process obtains any items from any other process. This is clearly unsustainable in the presence of two or more Byzantine processes because one cannot prevent two Byzantine processes from conspiring in order for one of them to obtain the item of the second one. A simple but flawed fix would be to modify this definition as follows: if any correct process does not obtain its item, then no process obtains any items from any *correct* process. If it first seems correct, this definition of *fairness* now allows a correct process to obtain the item of a Byzantine process, even if other correct processes do not obtain anything.

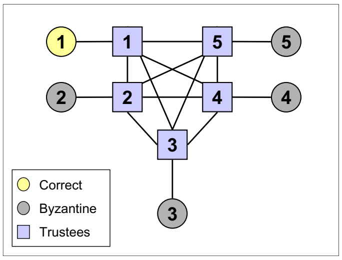
Coming back to our specification, *integrity* ensures that no process obtains an item offered by a correct process and matching the description of some other correct process. Notice that this does not prevent a Byzantine process from illicitly obtaining the item destined to or offered by some other Byzantine process, since such a behavior cannot be prevented and does not prejudice any correct process. Then, *fairness* guarantees that if any process obtains its desired item offered by some other process, with at least one of them being correct, then every correct process also obtains its desired item. In other words this property prevents a Byzantine process from taking advantage of a correct process but does not protect other Byzantine processes from their own incorrect behaviors. More trivially, it also ensures that no correct process takes advantage of any process.

3 The *Reachable Majority* Condition

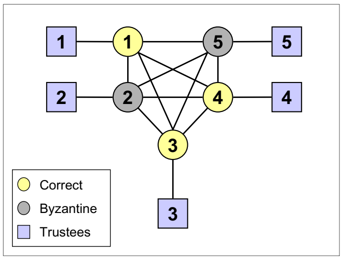
In a companion paper [4], we showed that a *necessary* condition to solve fair exchange in the model with trustees is to have every correct participant reliably connected to a majority of trustees. To formally define this condition, named the *reachable majority* (RM) condition, we must first define the notion of *reliable path* as follows. Let p_i and p_j be two correct processes of Π^+. We say that p_i and p_j are connected by a *reliable path*, if there exists at least one path between p_i and p_j such that no process along that path is Byzantine. The RM condition is then formally defined as follows.

Definition 1 (Reachable majority condition). *Topological condition under which, for any correct process $p \in \Pi$ and any failure pattern $f \in F_b$, p is connected by a reliable path to a strict majority of trustees, i.e., $\lfloor \frac{n}{2} + 1 \rfloor$, even in the presence of up to b Byzantine processes.*

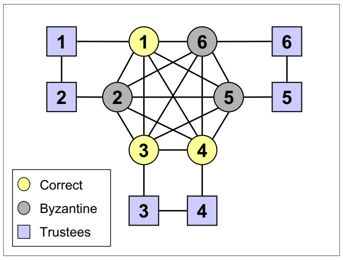
Note that trustees described in Definition 1 are called *major* trustees, whereas others are called *minor* trustees. The strict majority of Definition 1 ensures that the set of major trustees is identical for all correct processes, since if two processes have a single major trustee in common, then they have all their major trustees in common. The main focus of this paper is to show that not only is this condition necessary, it is also sufficient (Theorems 1 and 2 hereafter). This condition then allows us to better reason on the solvability of fair exchange and to compare different topologies. Indeed, given a topology and a number of Byzantine processes, one can infer whether a solution exists in that context. Or maybe more interestingly, it is possible to determine the maximum number of Byzantine processes that a specific network topology may sustain and yet still allow true fair exchange (by opposition to probabilistic fairness). Note that if the RM condition is met, it implies that all correct processes and a majority of trustees are interconnected by reliable paths. However, it is important to note that it does not imply, nor require, a majority of correct processes. Figure 2 gives examples of topologies allowing true fair exchange, including their respective upper bounds on the number of Byzantine processes. As illustrated in Figure 2(a), a TTP is able to sustain any number of Byzantine processes, whereas Figure 2(b) and (c) show topologies sustaining respectively a minority of Byzantine processes and up to the parity between correct and Byzantine processes.



(a) $b \leq n-1$



(b) $b \leq \lceil \frac{n}{2} - 1 \rceil$



(c) $b \leq \frac{n}{2}$

Fig. 2. Topologies allowing true fair exchange

3.1 Impossibility and Solvability Theorems

As already mentioned, in [4] we showed that the RM condition is a *necessary* condition in order to deterministically solve fair exchange in the model with trustees. Hereafter, Theorem 1 gives an informal reading of this result. Furthermore, in this paper, we argue through Theorem 2 that the RM condition is also *sufficient* in order to solve fair exchange.

Theorem 1 (Impossibility). *In the context of a synchronous model with trustees and Byzantine failures, there is no deterministic solution to the fair exchange problem, if the reachable majority condition is not satisfied.*

The formal proof of Theorem 1 falls outside the scope of this paper and can be found in [4]. However, to give an intuition of its correctness, first observe that, in order to ensure fairness, trustees must make a *consistent* decision whether to allow their respective processes to obtain their items or not. Then, since a process and its trustee are directly connected, if some correct process p is not reliably connected to a majority of trustees, neither is its trustee p'. So, either p' is not allowed to make a decision and p ends up violating the termination property of fair exchange; or p' is indeed allowed to make a decision in the absence of a majority, in which case there is no means to prevent fairness from being violated, e.g., if another group of reliably connected trustees make a contradictory decision.

Theorem 2 (Solvability). *In the context of a synchronous model with trustees and Byzantine failures, there exists a deterministic solution to the fair exchange problem under the reachable majority condition.*

Proof. In Section 4, we present Algorithms 1 and 2, which combine to produce a generic solution to fair exchange, for any topology and any number of Byzantine processes respecting the RM condition. We then prove the correctness of Theorem 2 by proving that our solution preserves the *validity*, *uniqueness*, *non-triviality*, *termination*, *integrity* and *fairness* properties of fair exchange.

4 Fair Exchange Under the RM Condition

In this section, we propose a solution to fair exchange that relies on the use of trustees. As described in Section 2.1 (system model), participants communicate by message passing and the network is a connected graph with respective participants and trustees connected directly. Our solution is composed of Algorithm 1 and Algorithm 2 and, other than *perfect links* presented in the model, they rely on two communication modules described hereafter, i.e., a *best-effort multicast* module and a *Byzantine agreement* module. Note that we merely aim at proving that a generic solution does exist under the RM condition and are thus not concerned with performance.

Best-Effort Multicast (BM). In order to solve fair exchange, Algorithms 1 and 2 rely on a *best-effort multicast* module that provides processes (participants and trustees) with the means to send messages to any group of processes with best effort. As described in our model, directly connected processes communicate via reliable channels. However, two processes that do not benefit from a direct link are not necessarily reliably connected, since paths between them might go through a Byzantine process, making communications potentially unreliable. The BM module provides a means to reliably send messages to processes accessible through at least one reliable path.[4] The module provides two primitives, *send* and *deliver*, described hereafter.

BM.send($p_i, S,$ 'TYPE'$, m$) – Enables a process p_i to multicast a message m to a defined set S of processes. The message type prevents confusion among different messages.

BM.deliver($p_i, p_j,$ 'TYPE'$, m$) – Works as a callback and enables a process p_j of S to receive a message m from process p_i.

Hereafter, we present the *validity* and *agreement* properties of best-effort multicast, which are the two main properties ensured by the BM module. A *no creation* property is also part of the specification of the best-effort multicast abstraction but is not detailed here. Note that one can ensure that the *validity* property of BM is achieved within some maximum time bound Δ_{BM}, e.g., by having $\Delta_{BM} = n \times \Delta_{PL}$ in the worst case.

Validity. Let p_i and p_j be any two correct processes connected by a reliable path, if p_i BM.sends a message m to a set S, with $p_j \in S$, then p_j eventually delivers m.

Agreement. Let p_i and p_j be any two correct processes of some set S that are connected by a reliable path, if p_i BM.delivers a message m BM.sent to S, then p_j BM.delivers m.

Byzantine Agreement (BA). In Algorithm 2, we use a Byzantine Agreement module that provides trustees with a means to reach agreement among major trustees, in spite of Byzantine failures that may occur along the various paths. This version of Byzantine agreement is largely based on [13]. However it differs in the sense that no Byzantine processes participate in the agreement (only trustees) but communications along unreliable paths may be blocked by Byzantine processes. Now, by considering minor trustees responsible for Byzantine failures happening along the unreliable paths leading to them, one can then apply Byzantine agreement to our model, i.e., by considering major trustees as correct processes, minor trustees as Byzantine processes and unreliable paths as reliable. The BA module provides three primitives, BA.start(), BA.send() and BA.deliver(), described hereafter in details.

BA.start(p'_j) – Enables a trustee p'_i to start an execution of BA in order to receive a message from a trustee p'_j. For each execution of the protocol, every trustee calls the *start* primitive at the same time (see explanation below) and trustee p'_j calls the *send* primitive.

BA.send(p'_i, m) – Enables a trustee p'_i to reliably broadcast a message m to all trustees.

[4] This can be achieved using flooding as presented in Appendix A or some more sophisticated algorithm [12].

BA.deliver(p'_i, M) – Works as a callback and enables a trustee p'_j to receive a set M of messages as the result of a reliable broadcast by trustee p'_i. Possible outcomes of the broadcast are twofold: (1) M is a singleton, meaning that transmissions from the sender were not blocked, so that message can be used; (2) M is the empty set, meaning that the sender did not call the *send* primitive in a timely fashion or that messages from the sender were blocked.

Intuitively, the goal consists in preventing Byzantine processes along unreliable paths from causing major trustees to receive different sets M. When relying on unforgeable signed messages, a solution is known to exists for any number of Byzantine processes [13], i.e., in our case minor trustees. Hereafter, we recall the two interactive consistency (IC) properties ensured by the BA module.

IC1 – Agreement. If a major trustee BA.delivers a set of messages M, then every major trustee BA.delivers M.

IC2 – Validity. If a major trustee BA.sends a message m, then every major trustee *eventually* BA.delivers the set $\{m\}$.

An implicit assumption in [13] is that all trustees roughly start at the same time to allow the absence of messages to be detected. Since all trustees roughly start Algorithm 2 at the same time, the *start* primitive of BA enables us to explicitly ensure this assumption by having all trustees calling the primitive at the same time (line 13 of Algorithm 2), i.e., at time $t_0 + \Delta_{BM}$. This ensures *termination* of BA, even if a trustee does not send any vote or messages are blocked by some Byzantine processes.

4.1 Fair Exchange Algorithm

Algorithms 1 and 2 provide a generic solution to the fair exchange problem for any topology and any number of Byzantine processes meeting the RM condition. For sake of simplicity, we assume that all correct processes – including all trustees – have local clocks that are synchronized within some fixed maximum error, as discussed in [14], so they are able to start the algorithms at the same time. We also assume that *upon* actions are executed atomically with respect to one another. Participants execute Algorithm 1, which initiates the fair exchange protocol, and trustees execute Algorithm 2. In Algorithm 1, each participant sends an encrypted version of its offered item to the trustee of the corresponding participant, according to Ω (the terms of the exchange). It then waits to receive and release the content of the first message sent by its trustee. The termination of Algorithm 1 is ensured by the timeout contained in Algorithm 2. In Algorithm 2, each trustee waits to receive the item expected by its associated participant. Algorithm 2 is then structured in two phases described hereafter: (1) the voting phase, and (2) the clue exchange phase.

Voting Phase. In this phase, trustee p'_i sends its vote to every trustee to inform them that it holds the expected item, and waits to receive the vote of every trustee. In Algorithm 2, once trustee p'_i receives the encrypted item (line 7), it

Algorithm 1. Fair Exchange executed by participant p_i, with $(p_i, p_j) \in \Omega$

```
1: Uses: Perfect Link (PL), Best-effort Multicast (BM)
2: Initialisation:
3:    released ← 'false'

4: function offer(item, $p_j$)
5:    BM.send($p_i$, $\{p'_j\}$, 'ITEM', encrypt($p'_j$, item))    {sends its encrypted item to $p'_j$}

6: upon PL.deliver($p'_i$, $p_i$, item) do                      {callback from PL}
7:    if ¬released then                                         {avoids releasing}
8:       released ← 'true'                                      {more than once}
9:       release(item)                                          {releases the item received}
```

deciphers it using its private key, checks if it matches its description and starts the voting process. The trustee signs and broadcasts its PROCEED vote (line 12) using BA, indicating that it holds the expected item. It also starts BA for each trustee to ensure termination of all executions of BA. Then, upon reception of a vote, the validProceedVote() function checks if the delivered set is a singleton containing the PROCEED vote of the sender (line 17). If the vote is valid, it is added to the set of votes. Once all votes are gathered, a trustee knows that every trustee voted PROCEED and that they thus hold the expected item. With that information, trustee p'_i enters the final phase by signing and then sending the n votes – called the i-th clue – to every trustee (line 21).

Clue Exchange Phase. In this phase, trustee p'_i sends its clue to all trustees to inform them that it received all n votes, and waits to receive the clues from a majority of trustees (line 27). Upon reception of a clue, the validClue() function checks if the clue contains a signed set of all n PROCEED votes (line 25). With $\lfloor \frac{n}{2} + 1 \rfloor$ clues, it sends the deciphered item to its corresponding participant (line 28). The majority is necessary to ensure that at least one major trustee was able to produce its i-th clue in order for any process to release its item. At this stage, no Byzantine process is able to prevent trustees of correct processes to send the expected item to their respective process.

4.2 Correctness Proof

In the following, we prove that Algorithms 1 and 2 solve fair exchange under the *reachable majority* condition. Our correctness proof shows that Algorithms 1 and 2 preserve the *validity*, *uniqueness*, *non-triviality*, *termination*, *integrity* and *fairness* properties of fair exchange. Based on Lemma 1, the respective theorems hereafter validate each of these properties. Note that, hereafter, the term *process* is only used to designate participants, i.e., processes of Π, unless specifically mentioned otherwise.

Algorithm 2. Fair Exchange protocol executed by trustee p'_i

```
Uses: Perfect Link (PL), Best-effort Multicast (BM), Byzantine Agreement (BA)
Initialisation:
    $t_0 \leftarrow \mathsf{time}()$                                   {sets $t_0$ to starting time}
    $d_i \leftarrow ...$                                               {sets description to known value}
    $\mathsf{item} \leftarrow \bot$                                    {sets variable to null}
    $\mathsf{votes}, \mathsf{clues} \leftarrow \emptyset$              {sets variables to empty set}

upon $\mathsf{BM.deliver}(p_j, p'_i, \text{‘ITEM’}, \mathsf{sealedItem})$ do     {callback from BM}
    if $(\mathsf{item} = \bot)$ then                                   {checks for duplicate send}
        $\mathsf{item} \leftarrow \mathsf{decipher}(\mathsf{sealedItem})$    {deciphers and stores received item}
        if $\mathsf{desc}(\mathsf{item}) = d_i$ then                   {check if item matches description}
            $\mathsf{vote} \leftarrow \mathsf{sign}(\text{‘PROCEED’})$     {produces PROCEED vote}
            $\mathsf{BA.send}(p'_i, \mathsf{vote})$                    {sends vote}

upon $time() > t_0 + \Delta_{BM}$ do                                   {item exchange phase is over}
    for all $p'_j \in \Pi'$ do                                         {for all trustees}
        $\mathsf{BA.start}(p'_j)$                                      {starts BA}

upon $\mathsf{BA.deliver}(p'_j, \mathsf{vote})$ do                     {callback from BA}
    if $\mathsf{validProceedVote}(\mathsf{vote})$ then                 {checks vote}
        $\mathsf{votes} \leftarrow \mathsf{votes} \cup \mathsf{vote}$  {adds $p'_j$'s vote to set}
        if $(|\mathsf{votes}| = n)$ then                               {if all votes are PROCEED}
            $\mathsf{clue} \leftarrow \mathsf{sign}(\mathsf{votes})$   {produces clue}
            $\mathsf{BM.send}(p'_i, \Pi', \text{‘CLUE’}, \mathsf{clue})$   {sends clue}
    else
        $\mathsf{PL.send}(p'_i, p_i, \varphi)$                         {sends $\varphi$ to $p_i$}

upon $\mathsf{BM.deliver}(p'_j, p'_i, \text{‘CLUE’}, \mathsf{clue})$ do   {callback from BM}
    if $\mathsf{validClue}(\mathsf{clue})$ then                        {checks if message is valid}
        $\mathsf{clues} \leftarrow \mathsf{clues} \cup \{\mathsf{clue}\}$   {adds $p_j$'s clue to set}
        if $(|\mathsf{clues}| > n/2)$ then                             {checks for majority of clues}
            $\mathsf{PL.send}(p'_i, p_i, \mathsf{item})$               {sends item to $p_i$}
```

Lemma 1. *If some trustee does not receive the expected encrypted item, then no trustee sends an item at line 28 of Algorithm 2.*

Proof. If some trustee does not receive the expected item, it does not send the PROCEED vote. Hence no trustee receives all n PROCEED votes, so no trustee sends its i-th clue. If no trustee sends its i-th clue, then no trustee receives any clue. Without a majority of clues, no trustee sends the item to its corresponding participant at line 28 of Algorithm 2.

Theorem 1 (Validity). *If a correct process p_i releases an item x, then either $x \in M$ and x matches d_i, or x is the abort item φ.*

Proof. In Algorithm 1, a process p_i only releases an item at line 9. Process p_i releases upon reception of an item from its trustee p'_i, so the possible items are those sent by p'_i in Algorithm 2. In Algorithm 2, trustee p'_i explicitly sends the abort item φ at line 23 so p_i would release φ. The only other case of item transmission is at line 28: p'_i sends the item that is stored in variable item. From Lemma 1, if a trustee sends an item at line 28, it has previously received the expected item and stored it in variable item. Since, from line 8, no two different items can be stored in variable item, p'_i sends the expected item at line 28. So p_i would release the expected item.

Theorem 2 (Uniqueness). *No correct process releases more than once.*

Proof. The boolean variable released in Algorithm 1 and the atomic execution of upon statements prevent any correct process from releasing more than once.

Theorem 3 (Non-triviality). *If all processes are correct, no process releases the abort item φ.*

Proof. Since every process is correct, every process sends the correct encrypted item at line 5 of Algorithm 1 as agreed in the terms of the exchange. From the *validity* property of BM, every trustee p'_i receives an item matching description d_i before time $t_1 = t_0 + \Delta_{BM}$, so every trustee produces and sends its PROCEED vote at line 12 of Algorithm 2 in a timely fashion. From the IC2 property of BA, no process receives an invalid PROCEED vote. So finally, no trustee sends the abort item φ (line 23) of Algorithm 2 and thus no process releases φ.

Theorem 4 (Termination). *Every correct process* eventually *releases an item.*

Proof. The assumption that participants and trustees start Algorithms 1 and 2 at the same time and the timeout at line 13 of Algorithm 2 ensures that every trustee starts all n executions of BA at the same time. This implies that, from the existence of a time bound for the termination of BA and the IC1 property, there is a time after which: either every trustee of correct processes receives at least one invalid PROCEED vote and sends the abort item φ, prompting the corresponding correct process to release φ; or every major trustee receives all n valid PROCEED votes. In the latter case, every major trustee produces and sends its i-th clue at line 21 of Algorithm 2. From the *validity* property of BM and the *reachable majority* condition, every trustee of correct processes receives a majority of clues and then sends the item at line 28 of Algorithm 2. Finally, from the *termination* property of PL, every correct process releases the item.

Theorem 5 (Integrity). *No process p_j releases an item m_i, with process p_i correct, if m_i matches description d_k of some correct process p_k, with $p_k \neq p_j$.*

Proof. Firstly, since any process p_k and its trustee p'_k are directly connected, no process p_j intercepts the transmission of any deciphered item m_i by p'_k at line 28 of Algorithm 2. Secondly, only in a single step of Algorithm 1, i.e., at line 5, does a correct process p_i transmit its item m_i through the network. Since p_i is correct, p_i encrypts m_i using the public key of p'_k in order to send it through the network. So no process other than p_i and p_k holds a deciphered version of

m_i and, since both are correct, they do not send a deciphered version of m_i to p_j. From assumption on the PKI unforgeability, p_j is not capable of obtaining a deciphered version of m_i and thus does not release m_i.

Theorem 6 (Fairness). *If any process p_i releases an item m_j matching description d_i, with p_i or p_j correct, then every correct process p_k releases an item matching description d_k.*

Proof. The proof is by contradiction.

Assume that some correct process p_k does not release an item matching description d_k and that some other process p_i releases an item m_j matching description d_i, with p_i or p_j correct. If p_i releases m_j (line 9 of Algorithm 1), either p_i is correct and only releases an item received from its trustee p'_i; or p_j is correct and encrypted m_j before sending it to p'_i (line 5 of Algorithm 1) and thus p_i is only capable of releasing m_j by receiving it from its trustee p'_i. So in either cases, if p_i releases m_j, m_j is received from trustee p'_i, which sends m_j at line 28 of Algorithm 2. Trustee p'_i thus receives a majority of clues in some previous steps. From the *reachable majority* condition, at least one of these clues is produced by some major trustee p'_x. Trustee p'_x thus receives all n PROCEED votes. So, from the IC1 property of BA, every major trustee also receives all n PROCEED votes, including all trustees of correct processes. This implies that no trustee of correct processes sends the abort item φ (line 23 of Algorithm 2), including p'_k, so p_k does not release φ. From the validity and termination properties of FE, if p_k does not release φ, then p_k releases an item matching description d_k. A contradiction.

4.3 Discussion

As presented in Section 4, our generic solution relies both on best-effort multicast (BM) and Byzantine agreement (BA) modules. The BM module is used in Algorithms 1 and 2, i.e., by participants and trustees, whereas the BA module is only used in Algorithm 2, i.e., by trustees.

Since both modules share very similar *validity* and *agreement* properties, a reasonable question is: could we have done with only one module? The answer is: yes, a modified version of BM would be sufficient. To understand why, let us first point out a key guarantee offered by BA: trustees always *eventually* deliver a set of messages, even if the sender did not call the send primitive or all its messages where blocked by Byzantine processes (in which case the set is empty). The eventual delivery of BA is achieved through the use of the *start* primitive, allowing trustees to detect the absence of messages.[5]

The BM module, on the contrary, offers no such guarantee. However, by adding a *start* primitive to BM and by slightly changing its semantics, we could rely on the BM module to reach deterministic agreement among major trustees.

[5] Major trustees are thus able to agree on the votes of all trustees, including minor trustees, even if some (or all) messages sent by minor trustees are blocked.

However, since the BM module already accomplishes two different tasks, i.e., point-to-point and multicast communication, overloading it with a third semantics would make our solution more difficult to understand.

5 Related Work

Research on fair exchange has produced an impressive body of work over the past decades, as testified by several surveys [15,16]. At least three different communities of researchers are showing major interest in this problem, namely people active in *e*-business solutions, in cryptographic algorithms and in distributed systems. This diversity results in a variety of problem statements and underlying assumptions, as well as an even larger number of approaches to solve fair exchange.

Modeling Fair Exchange. The fair exchange problem comes basically in two flavors, namely a *weak* variant and a *true* variant [16]. Weak fair exchange does not require the exchange to be fair but rather that honest peers are able to gather evidence of potential misbehaviors. This variant thus assumes that misbehaving peers can be brought to justice, which is not the case in our approach. The problem we address in this paper is true fair exchange, which on the contrary requires a strong enforcement of fairness.

Within the realm of true fair exchange, various specifications have been proposed, with slightly different sets of properties [15]. Among these properties, fairness is the most difficult to capture and hence where most specifications tend to differ, as in [2,5,11]. Despite what is sometimes claimed, several such specifications are really meaningful for exchanges involving only two processes, i.e., they are impossible to satisfy in models allowing more than one Byzantine process. Note also that many researches explicitly aimed at the fair exchange variant involving only two peers [5,17,18,19,20], in particular when it comes to specific applications of fair exchange, e.g., exchanges of digital signatures, of emails and their receipts, etc. Our specification of the fair exchange problem, on the contrary, considers the general case where more than two peers might be involved, as already discussed in Section 2.2.

Besides proposing a specification, some authors also discuss the difficulty of fair exchange and propose impossibility results in various models. In [11], fair exchange is measured against consensus, and an impossibility result on fair exchange in asynchronous models is shown by comparison with the FLP impossibility [21]. In [22], fair exchange is shown to be impossible to solve *deterministically* in an asynchronous system with no Trusted Third Party (TTP). In another feasibility study [23], complex exchanges are broken into sub-exchanges – each relying on a different TTP – and represented as a graph. Reduction rules are then applied to the graph in order to demonstrate the feasibility of the exchange. This method also makes it possible to illustrate how closely exchange feasibility relies on trust. Along that line, we have shown that fair exchange is insolvable in a synchronous model in the absence of some identified process that every other process can trust *a priori* [4].

Before moving to the discussion on existing fair exchange solutions, let us clarify an often misunderstood specificity of fair exchange. Indeed, this misunderstanding leads some people to believe that the above impossibility results are in fact contradicting an important result by Chaum et al. in [24]. Intuitively, this result states that any multiparty protocol can be achieved in an unconditionally secure manner, provided that the system is synchronous and that at least $\frac{2}{3}$ of the peers are honest. The key difference here is in what one is really trying to achieve. Indeed, the problem considered by Chaum et al. consists in having a set of peers compute a multiparty function while preserving *privacy* regarding each peer's input and output [25]. However, fairness is out of their scope, i.e., they do not achieve it, nor discuss it, hence the confusion, since the absence of discussion may unintentionally lure the reader into thinking otherwise.

Solving Fair Exchange. Most solutions to fair exchange rely on some kind of Trusted Third Party (TTP). A TTP is a process directly accessible to all processes. Fairness is thus trivially ensured by having processes send their items to the TTP, which forwards the items, if the terms of the exchange are fulfilled [26]. A TTP brings synchronism and control over terms of the exchange in order to ensure fairness but constitutes a bottleneck and a single point of failure. For this reason, various so-called *optimistic* algorithm have been proposed that only involve the TTP when something goes wrong, i.e., when an attempt to cheat is detected [5,19,26,27,28]. However *optimistic* approaches are based on the strong assumption that the environment is mostly honest. To weaken the role of the TTP, in [18] for instance, Franklin and Reiter propose a solution using a *semi-trusted* third party that can misbehave on its own but does not conspire with either of the two participant peers. Similarly, the authors of [29] propose a solution based on a cluster of untrusted servers acting as third parties. In the latter paper, however, the authors recognize that they are merely solving a variant of the weak fair exchange.

By relying on fully decentralized tamperproof modules, other approaches depart from the traditional TTP-based approach [1,2,3], assuming fully connected processes but embedded tamperproof modules dependant of their process for communicating. A very interesting feature of the approach proposed in [2] lies in its ability to gracefully degrade its quality of service from true fairness to probabilistic fairness.

6 Concluding Remarks

In this paper, we extended a previous result [4] by proposing a necessary and sufficient topological condition – the *reachable majority* condition – on the solvability of fair exchange in a synchronous model with Byzantine failures and trustees. We gave a solution to fair exchange under the *reachable majority* condition, along with its correctness proof. This result thus validates the correctness of our *reachable majority* condition. Currently, we are further studying the relationship between various topologies and the *reachable majority* condition. We are also further investigating the relationship between our results and those found in the domain of *secure multiparty computation* and *fair computation* [30]. In [24] for instance, Chaum et al. show that any multiparty protocol can be achieved in an unconditionally secure manner, provided that the system is synchronous and that at least $\frac{2}{3}$ of the peers are honest.

Acknowledgements

This research is partly funded by the Swiss National Science Foundation, in the context of Project number 200021-104488.

References

1. Avoine, G., Gärtner, F., Guerraoui, R., Kursawe, K., Vaudenay, S., Vukolic, M.: Reducing fair exchange to atomic commit. Technical report, Swiss Federal Institute of Technology (EPFL) (2004)
2. Avoine, G., Gärtner, F., Guerraoui, R., Vukolic, M.: Gracefully degrading fair exchange with security modules (extended abstract). In: In Proceedings of the 5th European Dependable Computing Conference - EDCC 2005. (2005)
3. Avoine, G., Vaudenay, S.: Fair exchange with guardian angels. Technical report, Swiss Federal Institute of Technology (EPFL) (2003)
4. Garbinato, B., Rickebusch, I.: Impossibility results on fair exchange. In: Proceedings of the 6th International Workshop on Innovative Internet Community Systems (I2CS'06). Volume LNCS., Springer (2006)
5. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures. IEEE Journal on Selected Area in Communications **18** (2000) 593–610
6. Dyer, J., Lindemann, M., Perez, R., Sailer, R., van Doorn, L., Smith, S., Weingart, S.: Building the IBM 4758 secure coprocessor. Computer **34**(10) (2001) 57–66
7. Bajikar, S.: Trusted Platform Module (TPM) based Security on Notebook PCs – White Paper. Intel Corporation – Mobile Platforms Group. (2002)
8. Doudou, A., Garbinato, B., Guerraoui, R.: Tolerating Arbitrary Failures with State Machine Replication. In: Dependable Computing Systems: Paradigms, Performance Issues, and Applications. Wiley (2005) 27–56
9. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM **43**(2) (1996) 225–267
10. Hadzilacos, V., Toueg, S.: Fault-tolerant broadcasts and related problems. (1993) 97–145
11. Pagnia, H., Gärtner, F.: On the impossibility of fair exchange without a trusted third party. Technical report, Swiss Federal Institute of Technology (EPFL) (1999)
12. Drabkin, V., Friedman, R., Segal, M.: Efficient byzantine broadcast in wireless ad-hoc networks. In: DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05), Washington, DC, USA, IEEE Computer Society (2005) 160–169
13. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems **4**(3) (1982) 382–401
14. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM **27**(2) (1980) 228–234

15. Markowitch, O., Gollmann, D., Kremer, S.: On fairness in exchange protocols. In: Proceedings of the 5th International Conference Information Security and Cryptology (ICISC 2002). Volume 2587 of Lecture Notes in Computer Science., Springer (2002) 451–464
16. Ray, I., Ray, I.: Fair exchange in e-commerce. SIGecom Exchanges **3**(2) (2002)
17. Ateniese, G.: Efficient verifiable encryption (and fair exchange) of digital signatures. In: CCS '99: Proceedings of the 6th ACM conference on Computer and communications security, New York, NY, USA, ACM Press (1999) 138–146
18. Franklin, M., Reiter, M.: Fair exchange with a semi-trusted third party (extended abstract). In: CCS '97: Proceedings of the 4th ACM conference on Computer and communications security, New York, NY, USA, ACM Press (1997) 1–5
19. Micali, S.: Simple and fast optimistic protocols for fair electronic exchange. In: PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing, New York, NY, USA, ACM Press (2003) 12–19
20. Ray, I., Ray, I., Natarajan, N.: An anonymous and failure resilient fair-exchange e-commerce protocol. Decision Support Systems **39**(3) (2005) 267–292
21. Fischer, M., Lynch, N., Paterson, M.: Impossibility of Distributed Consensus with One Faulty Process. J. ACM **32** (1985) 374–382
22. Even, S., Yacobi, Y.: Relations among public key signature systems. Technical report, Technion - Israel Institute of Technology (1980)
23. Ketchpel, S., García-Molina, H.: Making trust explicit in distributed commerce transactions. In: Proceedings of the International Conference on Distributed Computing Systems. (1995)
24. Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: STOC '88: Proceedings of the 20th ACM symposium on Theory of computing, New York, NY, USA, ACM Press (1988) 11–19
25. Goldreich, O.: The Foundations of Cryptography. Volume 2. Cambridge University Press (2004)
26. Bürk, H., Pfitzmann, A.: Value exchange systems enabling security and unobservability. Computers & Security **9**(9) (1990) 715–721
27. Bao, F., Deng, R.H., Mao, W.: Efficient and practical fair exchange protocols with off-line TTP. In: RSP: 19th IEEE Computer Society Symposium on Research in Security and Privacy. (1998)
28. Baum-Waidner, B., Waidner, M.: Round-optimal and abuse free optimistic multiparty contract signing. In: Automata, Languages and Programming. Number 1853 in Lecture Notes in Computer Science (LNCS), Springer (2000) 524–535
29. Srivatsa, M., Xiong, L., Liu, L.: Exchangeguard: A distributed protocol for electronic fair-exchange. In: 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), IEEE Computer Society (2005)
30. Goldwasser, S., Levin, L.: Fair computation of general functions in presence of immoral majority. In: CRYPTO '90: Proceedings of the 10th International Cryptology Conference on Advances in Cryptology, London, UK, Springer-Verlag (1991) 77–93

A Best-Effort Multicast

Algorithm 3 provides a solution to the *best-effort multicast* abstraction presented in Section 4 and thus shows that the BM module is implementable in the context of our model. We assume that every process knows its direct neighbors and we define V_{p_i} as the set of neighbors of process p_i. Intuitively Algorithm 3 satisfies the properties of best-effort multicast by having correct processes flooding the network with the message. Flooding is achieved by forwarding any received message the first time it is received. Upon reception of that message, if the process is included in the set S of recipients, it also delivers the message. Note that having a Byzantine process deliver a message it was not suppose to does not jeopardize the validity of BM, nor cause any sort of problems.

Algorithm 3. Best-effort multicast protocol executed by process p_i

```
 1: Uses:
 2:    Perfect Link (PL)

 3: Initialisation:
 4:    forwarded ← ∅                                          {set of forwarded messages}

 5: function send($p_i, S, m$)
 6:    for all $p_j \in V_{p_i}$ do                                     {for all neighbors}
 7:       PL.send($p_i, p_j, \langle p_i, S, \mathsf{sign}_i(m)\rangle$)             {sign and send the message}
 8:    if $p_i \in S$ then                                  {check if message destined to self}
 9:       deliver($m$)                                            {deliver the message}

10: upon PL.deliver($p_j, p_i, \langle p_k, S, \mathsf{sign}_k(m)\rangle$) do
11:    if $m \notin$ forwarded then                                   {check if not forwarded}
12:       forwarded ← forwarded ∪ $\{m\}$                  {add the message to forwarded set}
13:       for all $p_x \in V_{p_i} - \{p_j\}$ do                      {for all neighbors except $p_j$}
14:          PL.send($p_i, p_x, \langle p_k, S, \mathsf{sign}_k(m)\rangle$)                  {forward the message}
15:       if $p_i \in S$ then                               {check if message destined to self}
16:          deliver($m$)                                         {deliver the message}
```

Correctness Proof. In the following, our correctness proof aims at showing that Algorithm 3 preserves the Agreement and Termination properties of best-effort multicast and that such a module is thus implementable in our model. Note that in Lemma 1 the term 'receive a message' does not imply that the message is delivered but it relates to messages that are either obtained from the send() function (line 5 of Algorithm 3) or from the PL.deliver() callback (line 10 of Algorithm 3).

Lemma 1. *Let p_i and p_j be any two correct processes that are connected through a reliable path, if p_i receives a message m, then p_j receives m.*

Proof. The proof is by induction.
Basis step. Assume that some correct process p_i receives a message m (line 5 or 10) and that p_i and p_j are directly connected. So either p_i is the originator of m and sends m to processes of V_{p_i}; or p_i receives m from some process p_x and sends m to processes of $V_{p_i} - \{p_x\}$. In both cases, from the termination property of perfect links, all processes of V_{p_i} eventually receive m. From our initial assumption, since $p_j \in V_{p_i}$, p_j receives m.

Inductive step. Assume that any two correct processes p_i and p_j are connected through a reliable path. From definition of reliable paths, there exists a process p_k such that p_k is on that reliable path and $p_j \in V_{p_k}$. Moreover, p_k is correct and connected to p_i through a reliable path. So now assume that p_i and p_k receive a message m. Again either p_k is the originator of m and sends m to processes of V_{p_k}; or p_k receives m from some process p_y and sends m to processes of $V_{p_k} - \{p_y\}$. In both cases, from the termination property of perfect links, all processes of V_{p_k} eventually receive m. From our initial assumption, since $p_j \in V_{p_k}$, p_j receives m.

Theorem 1 (Agreement). *Let p_i and p_j be any two correct processes of S that are connected through a reliable path, if p_i delivers a message m, then p_j delivers m.*

Proof. Assume that any two correct processes p_i and p_j of S are connected through a reliable path and that p_i delivers a message m. So p_i receives m in a previous step of Algorithm 3 (line 5 or 10). From Lemma 1, p_j also receives m. Since p_j is a correct process of S, either m is in the forwarded set of p_j and p_j has delivered m, or m is not in the forwarded set of p_j and p_j delivers m at line 16.

Theorem 2 (Termination). *Let p_i and p_j be any two correct processes connected through a reliable path, with $p_j \in S$, if p_i sends a message m, then p_j eventually delivers m.*

Proof. Assume that any two correct processes p_i and p_j of S that are connected through a reliable path and that p_i sends a message m. So p_i receives m, as the originator of m. From Lemma 1, p_j also receives m. Since p_j is a correct process of S, either m is in the forwarded set of p_j and p_j has delivered m, or m is not in the forwarded set of p_j and p_j delivers m at line 16.