Gracefully Degrading Fair Exchange with Security Modules (Extended Abstract)

Gildas Avoine Felix Gärtner Rachid Guerraoui Marko Vukolić

Abstract. The *fair exchange* problem is key to trading electronic items in systems of mutually untrusted parties. In modern variants of such systems, 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 hosts are untrusted and could intercept and drop those messages. We describe a synchronous algorithm that ensures deterministic *fair exchange* if a majority of parties are honest, which is optimal in terms of resilience. If there is no honest majority, our algorithm degrades gracefully: it ensures that the probability of violating fairness can be made arbitrarily low. We prove that this probability is inversely proportional to the average complexity of the algorithm in terms of its number of communication rounds, and we supply the corresponding optimal probability distribution. Our algorithm uses, as an underlying building block, an early stopping subprotocol that solves, in a model with general omission failures, a specific variant of consensus we call *biased consensus*. Our modular approach contributes in bridging the gap between modern security (i.e., based on security modules) and traditional distributed computing (i.e., agreement with omission failures).

Category: Regular and student paper (Marko Vukolić and Gildas Avoine are full time students).

Author affiliations: G. Avoine, R. Guerraoui, M. Vukolić: EPFL, Switzerland; F. Gärtner: RWTH Aachen University, Germany.

Contact author: `Marko.Vukolic@epfl.ch`, EPFL, CH-1015 Lausanne, Switzerland; Tel: +41 21 693 81 23; Fax: +41 21 693 75 70.

Number of pages: 10 (without references and optional appendices).

Keywords: fair exchange; security; consensus; omission failures.

1 Introduction

1.1 Motivation

Fair exchange is a fundamental problem in systems with electronic business transactions. In fair exchange, the participating parties start with an item they want to trade for another item. They possess an executable description of the desired item, typically a boolean function with which an arbitrary item can be checked for the desired properties. Furthermore, they know from which party to expect the desired item and which party is expecting their own item. An algorithm that solves fair exchange must ensure that every honest party eventually either delivers its desired item or aborts the exchange (*termination* property). The abort option however is excluded if no party misbehaves and all items match their descriptions (*effectiveness*). The algorithm should also guarantee that, if the desired item of any party does not match its description, then no party can obtain any (useful) information about any other item (*fairness property*).

Fair exchange is easily solvable using a *trusted third party* through which all items can be exchanged [6]. The involvement of the trusted third party can be reduced using *optimistic* schemes where 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, distributed within all untrusted parties, as we explain below.

We consider in this paper a system where each party hosts a *security module* that is *tamper proof* (Fig. 1). Recently, manufacturers have begun to equip hardware with such modules: these include for instance smart cards or special microprocessors. Examples include the "Embedded Security Subsystem" within the recent IBM Thinkpad or the IBM 4758 secure co-processor board [7]. In fact, a large body of computer and device manufacturers has founded the Trusted Computing Group (TCG) [21] to promote this idea. Because their hardware is tamper proof, the software running within the security modules is certified and they can communicate through secure channels. In certain settings, the overall system can even assumed to be synchronous, i.e., it is reasonable to assume an upper bound on the relative speeds of honest parties (and their security modules) as well as on the communication delays between them. However, dishonest parties can still drop messages exchanged between the underlying security modules in order to violate the fairness of the exchange in their favor, i.e., obtain an item without giving away their own.

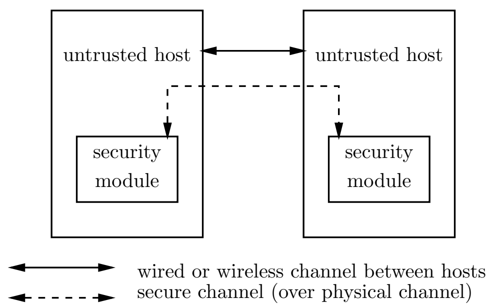

Fig. 1. Hosts and security modules.

The contribution of this paper is a synchronous distributed algorithm aimed at exchanging electronic items among untrusted parties, each hosting a security module. The algorithm provides the two following complementary features:

1. If a majority of parties is honest, then the algorithm deterministically guarantees the termination, effectiveness and fairness properties of fair exchange. This is optimal in terms of resilience: we indeed show that, even in a synchronous model with security modules, no deterministic algorithm solves fair exchange if half of the parties are dishonest.
2. If at least half of the parties turn out to be dishonest, then our algorithm degrades gracefully in the following sense. It still guarantees the termination and effectiveness properties of fair exchange, as well as ensures that the probability of violating fairness can be made arbitrarily low. We supply the probability distribution that optimizes the average complexity of the algorithm, in terms of its number of communication rounds, and we show that probability of violating fairness is inversely proportional to the average algorithm complexity.

Our algorithm is made of three phases, and we give the intuition underlying each phase below.

1. In the first phase, which we call the *initialization* phase, the security modules exchange the items that are supposed to be traded by their untrusted hosts. These items are not delivered by

the security modules to their untrusted hosts: this is only performed if the third phase (below) terminates *successfully*. Any security module can decide here to abort the exchange if some item is missing or does not match its expected description. The security module hosted by the party that initiates the exchange also selects here a *random number* k that it disseminates to all other security modules. The role of this random number is crucial in the second phase of the algorithm.

2. In the second phase, which we call the *fake* phase, all security modules exchange messages during k rounds; each round following the same communication pattern as in the third phase (below). The fact that the random number k, determined in the first phase, is not accessible to the untrusted parties is fundamental here. Roughly speaking, the goal of the *fake* phase is to make the probability, for *any* number of dishonest parties to successfully guess when the actual agreement phase is taking place (third phase below), arbitrarily low. If any dishonest party drops a message towards a honest party in this *fake* phase, the security module hosted by the latter simply aborts the exchange and forces other modules to abort the exchange as well, thus penalizing any dishonest host that might try to bias the exchange in its favor.
3. In the third phase, which we call the *agreement* phase, the security modules solve a problem we call *biased consensus*. In this problem, the processes (in our case the security modules) start from an initial binary value (a proposal) and need to decide on a final binary value: either to abort the exchange or commit it (and deliver the items to their untrusted hosts). Unlike in consensus [9], but like in NBAC (non-blocking atomic commit) [5, 20], the problem is biased towards 1: no process can decide 1 if some process proposes 0 (to avoid trivial solutions, the processes are supposed to decide 1 if no process fails or proposes 0). The agreement aspect of this problem is however different from consensus and NBAC; it is also biased towards 1: we simply require here that, if some process decides 1, then no correct process decides 0. We consider an early stopping algorithm that solves this problem in a model with general omissions, along the lines of [17].

Besides our main contribution, i.e., a new gracefully degrading fair exchange algorithm, our paper contributes in bridging the gap between security problems (fair exchange) and traditional distributed computing problems (consensus-like problems). We show indeed that deterministic fair exchange in a model with security modules is equivalent to biased consensus. By proving that biased consensus is impossible in a synchronous model [13] with general omission failures [18] if half of the processes can be faulty we directly establish a lower bound result for fair exchange in a model with tamper proof modules.

1.2 Roadmap

Section 2 defines our system model. Section 3 recalls the fair exchange problem, introduces biased consensus, and shows their equivalence in a model with security modules. We also state the impossibility of deterministic fair exchange without a honest majority which motivates our notion of gracefully degrading fair exchange. Section 4 describes our gracefully exchange fair exchange algorithm and states its correctness. Section 5 concludes the paper by discussing some related work. Most of the proofs are given in the optional appendices.

2 Model

The system we consider is composed of a set of processes, some modeling untrusted hosts and the other modeling security modules. These processes communicate by exchanging messages.

2.1 Untrusted hosts and security modules

More precisely, the set of processes we consider is divided into two disjoint classes: *untrusted hosts* (or simply *hosts*) and *security modules*. Two processes connected by a physical channel are said to be adjacent. 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): we say that P_A is *associated with* G_A. 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 of security modules, and the virtual communication links between them, the *security subsystem*. We call the part of the system consisting of hosts and the communication links between them the *untrusted system*. The notion of association can be extended to systems, meaning that, for any given untrusted system, the *associated security subsystem* is the system consisting of all security modules associated to any host in that untrusted system.

2.2 Security modules and virtual channels

Security modules are interconnected by a virtual communication network with bidirectional channels over the physical communication network among the hosts. For simplicity, we denote the participants processes (the security modules) by $G_1, \ldots, G_n$. We assume that between any two security modules G_i and G_j, the following properties are guaranteed: (1) Message contents remain secret from unauthorized entities; (2) If a message is delivered at G_j, then it was previously sent by G_i; (3) Message contents are not tampered with during transmission, i.e., any change during transmission will be detected and the message will be discarded; (4)If a message is sent by G_i to G_j and G_j is ready to receive the message, then the message will be delivered at G_j within some known bound Δ on the waiting time.

2.3 Trust and adversary model

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. A host may inhibit *all* communication between its associated security module and the outside world, yielding a channel in which messages can be lost.

A host *misbehaves* if it does not correctly follow the prescribed algorithm and we say that the host is *dishonest*. Otherwise it is said to be *honest*. Misbehavior is unrestricted (but computationally bounded as we pointed out). Security modules always follow their protocol, but since their associated hosts can inhibit all communication, this results in a system model of security modules with unreliable channels (the model of *general omission* [18], i.e., where messages may not be sent or received). In such systems, misbehavior (i.e., failing to send or receive a message) is sometimes termed *failure*. We call security modules associated with honest hosts *correct*, whereas those associated with dishonest hosts *faulty*. In a set of n hosts, we use t to denote a bound on the number of hosts which are allowed to misbehave and f the number of hosts which actually do misbehave ($f \leq t$). Sometimes we restrict our attention to the case where $t < n/2$, i.e., where a majority of hosts is assumed to be honest. We call this the *honest/correct majority assumption*.

Our model of the adversary is based on the strongest possible attack, the case in which all of the f dishonest hosts collude. We assume that adversary knows all the algorithms and probability distributions used.

3 Variations on Fair Exchange

In this section we recall the definition of fair exchange (FE), and we show that this problem, at the level of untrusted hosts, is in a precise sense equivalent to a problem that we call *biased consensus* (BC), at the level of the underlying security modules. Then, we state that biased consensus is impossible if half of the processes can be faulty and derive the impossibility of fair exchange if a majority of hosts can be dishonest (the detailed proof can be found in Appendix A). This motivates our definition of a weaker variant of fair exchange, the gracefully degrading FE.

3.1 Fair exchange

Definition 1 (Fair Exchange). *An algorithm solves* fair exchange *(FE) if it satisfies the following properties [1, 16]*

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

In case a host terminates without receiving the expected item, that host receives an abort indication (denoted $\perp$). The Timeliness property ensures that every honest host can be sure that at some point in time the algorithm 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.[1] Note that the first precondition of the Fairness property ("if the desired item of any host 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 expected description, which should clearly be considered unfair.

[1] 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].

3.2 Biased Consensus

Consider the following variant of consensus, we call *biased consensus* in a model where processes can fail by general omissions [18].

Definition 2 (Biased Consensus). *An algorithm solves* biased consensus *(BC) if it satisfies the following properties:*

- *(Termination) Every correct process eventually decides.*
- *(Non-Triviality) If no process is faulty or proposes 0, then no correct process decides 0.*
- *(Validity) No process decides 1 if some process proposes 0.*
- *(Biased Agreement) If any process decides 1, then no correct process decides 0.*

Processes invoke *biased consensus* using primitive $BCpropose(vote)$, $vote$ being a binary value, 0 or 1. Possible decisions are also 0 (*abort*) and 1 (*commit*). *Termination, Non-Triviality* and *Validity* are the same as in NBAC, whereas the *Biased Validity* is weaker than the *Agreement* property of NBAC [20]

We show below that FE and BC are equivalent in our model. To prove this claim, we first show that FE can be reduced to BC.

```
FairExchange(myitem, description, source, destination) returns item {
  ⟨send myitem to destination over secure channel⟩
  timed wait for ⟨expected item i from source over secure channel⟩
  ⟨check description on i⟩
  if ⟨check succeeds and no timeout⟩
  then vote := 1 else vote := 0 endif
  result := BCpropose(vote)
  if result = 1 then return i else return ⟨abort⟩ endif
}
```

Fig. 2. Using biased consensus to implement fair exchange: code of every host.

Theorem 1. *Biased consensus is solvable in the security subsystem, iff fair exchange is solvable in the associated untrusted system.*

Proof. (1) Assume that we have a solution to BC in the security subsystem consisting of security modules $G_1, \ldots, G_n$. Now consider the algorithm depicted in Fig. 2. This is a wrapper around the BC solution that solves FE at the level of the hosts. In other words, it is a reduction of FE into BC in our model. In the algorithm, a host hands to the associated security module its item and the executable description of the desired item, as well as identifiers of the hosts with which items should be exchanged. The security module exchanges the item with its partners, then checks the received item (*initialization* phase). Finally all security modules agree on the outcome using BC (*agreement* phase). The proposal value for BC is 1 if the check was successful and no abort was requested by the host in the meantime. If BC terminates with the process deciding 1, then the security module releases the item to the host. We now discuss each of the properties of fair exchange. The *Timeliness* property of FE is guaranteed by the *Termination* property of BC and the synchronous model assumption. Consider *Effectiveness* and assume that all participating hosts are honest and all items match their descriptions. All votes for BC will be 1. Now the *Non-Triviality* property of BC guarantees that all processes (recall that every process is correct) will decide 1 and subsequently return the item to their hosts. Consider now *Fairness* and observe that, in our adversary model, no dishonest host can derive any useful information from merely observing messages 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 process, then this process will engage in BC with a $vote = 0$ (note that this will happen even if the associated host is dishonest). *Validity* of BC implies that the exchange results in no process deciding 1, so none of the hosts receives anything from the exchange. Additionally, if some honest host receives nothing through the exchange, then the *Biased Agreement* property of BC implies that no host can receive anything.

(2) Conversely, BC can be implemented using FE by invoking

$$BCpropose(vote_i) = FairExchange(vote_i, 1, G_{i-1}, G_{i+1})$$

at every process G_i.[2]

Here we assume that if FE returns $\langle abort \rangle$ instead of the item, BC returns 0. So the votes, being exchange items in this case, are exchanged in a circular fashion among security modules. It is not difficult to see that FE properties guarantee the properties of BC. This is immediate for *Termination* and *Non-Triviality*. Consider now *Validity* and assume that G_j proposes 0 to BC. The item description checking at G_{j+1} will fail and the first part of FE *Fairness* ("If the desired item of any host does not match its description…") guarantees that every process that decides in BC decides 0. The second part of *Fairness* guarantees *Biased Validity*.[3] □

Theorem 2. *Consider a synchronous system where processes can fail by general omissions. No algorithm solves biased consensus if* $\lceil \frac{n}{2} \rceil$ *processes can be faulty.*

We give in optional Appendix A a detailed proof of this theorem, based on a partitioning of the processes. Note that this partitioning technique is different from the one traditionally used to prove consensus impossibility results (such as in [19]). In short, this is because, in consensus, if the processes of one partition all propose 0 and the processes of the other all propose 1, we clearly end up with two different decisions and a contradiction. This does not apply to biased consensus and the proof is slightly more involved. [4]

A direct corollary of the Theorems 1 and 2 leads to derive the following result:

Theorem 3. *Consider our model of untrusted hosts and security modules. No algorithm solves fair exchange if half of the hosts can be dishonest.*

3.3 Gracefully Degrading Fair Exchange

The impossibility of solving fair exchange (deterministically) if half of the processes can be dishonest, motivates the introduction of the following variant of the problem.

[2] To be precise, G_1 invokes $FairExchange(vote_1, 1, G_n, G_2)$ and G_n invokes $FairExchange(vote_n, 1, G_{n-1}, G_1)$

[3] Note that, in contrast to BC, FE satisfies an information-flow (i.e., security) property [15]. This is why it was necessary to argue about the special properties of security modules when reducing FE to BC and not vice versa.

[4] A similar difficulty arises from proving the impossibility for NBAC; indeed our result directly applies to NBAC which is stronger than biased consensus.

Definition 3 (Gracefully Degrading Fair Exchange). *An algorithm solves* gracefully degrading fair exchange *(GDFE) if it satisfies the following properties:*

- *The algorithm always satisfies the Timeliness and Effectiveness properties of fair exchange.*
- *If a majority of hosts are honest, then the algorithm also satisfies the Fairness property of fair exchange.*
- *Otherwise (if there is no honest majority), the algorithm satisfies Fairness with a probability* p $(0 < p < 1)$ *such that the probability of unfairness* $(1 - p)$ *can be made arbitrarily low.*

4 A Gracefully Degrading Fair Exchange Algorithm

4.1 Overview

Our GDFE algorithm is described in Figure 3. We assume that all processes involved in the algorithm know each other. The process with the lowest number is the initiator. We also assume synchronous communication model [13] in the security subsystem. Basically, our algorithm can be viewed as an extension of the algorithm of Figure 2, i.e., our reduction of deterministic fair exchange to biased consensus. However, whereas the algorithm of Figure 2 is made of an *initialization* phase followed by an *agreement* (BC) phase, the algorithm of Figure 3 introduces a *fake* phase between these two phases. This is the key to graceful degradation, i.e., to minimizing the probability of unfairness in the case when $t \geq n/2$. Basically, we do not run the BC algorithm immediately after the exchange of items (i.e., unlike in Fig. 2), but at some randomly picked round. In the meantime the processes exchange $fake$ messages and, if necessary, react to the behavior of hosts. If any process detects host misbehavior, i.e., a message omission, it aborts the algorithm immediately (line 15) and does not participate in BC.[5] It is important to notice that the underlying BC algorithm guarantees that no process decides 1 if some process does not participate in the algorithm (this missing process might have proposed 0). This is the way of penalizing any host that misbehaves in the first two phases of the algorithm.

The BC algorithm we use here is an adaptation of the early-stopping synchronous consensus algorithm of [17]. This algorithm solves BC if there is a majority of correct processes ($t < n/2$). It is early stopping in the sense that every process terminates in at most $min(f + 2, t + 1)$ rounds. This feature is important in minimizing the probability for the adversary to violate fairness (as we discuss in the next subsection): in short, the BC algorithm we consider has two vulnerable rounds: if the adversary misses them, the exchange terminates successfully. There are mainly two differences with the consensus algorithm of [17]: (1) the processes agree on a *vector* of initially proposed values rather then on a *set* of those (in the case of consensus); (2) we also introduce *dummy* messages to have a *full information protocol* [13], in order to have a uniform communication pattern in every round, as we explain below. (Our BC algorithm is given in Fig. 4 of Appendix B).

To make sure that the adversary has no means to distinguish the *fake* phase from the *agreement* phase (i.e., the BC algorithm), we make use of the same communication pattern in both phases, i.e., the same distribution and sizes of the exchanged messages: Every process sends a fixed-size message to every other process in every round, both in the *fake* phase and in the BC algorithm. Messages in $fake$ phase are, therefore, padded before sending, to the size of BC message. Hence, the adversary is not able to determine when BC starts, neither by observing when security modules send and receive messages, nor by observing the size of these messages.

[5] [14] uses a similar idea of choosing a random number of rounds and hiding it from the adversary to solve probabilistic non-repudiation (which is a special form of probabilistic fair exchange).

```
GDFairExchange(myitem, description, source, destination) returns item is

if ⟨G_i is initiator⟩ then % initialization phase - round 0
   ⟨pick a random number k according to a given distribution⟩
   foreach G_j ≠ destination do send (⊥, k) to G_j enddo
   send (myitem, k) to destination
else
   send (myitem, ⊥) to destination
endif
if ((item, ∗) has been received from source) and (check description on item succeeds) and
  ((∗, k) has been received from initiator) then
   vote_i := 1; k_i := k; item_i := item
else
   return (⊥)
endif

for round := 1, 2, ..., k_i do % fake phase - k rounds
   send ⟨padded vote_i⟩ to all
   if not((vote) has been received from all processes) then return (⊥) endif
enddo

vote_i := BCpropose(vote_i) % agreement phase - Biased Consensus
if (vote_i = 1) then return (item_i) else return (⊥) endif
end % of GDFairExchange
```

Fig. 3. Pseudocode of the Gracefully Degrading Fair Exchange algorithm: code of process G_i.

4.2 Correctness

Theorem 4. *The algorithm of Figure 3 solves gracefully degrading fair exchange.*

The detailed proof of Theorem 4 and other theorems of this section can be found in optional Appendix C. Here, we emphasize the case when the adversary controls half or more of the hosts. In addition, we define and give the optimal probability distribution of the random number k in this case.

If there is no honest majority, *Fairness* could be violated. However, this could occur only if the adversary successfully guesses in which round BC starts. Indeed, because our BC algorithm is *early stopping*, in order to succeed, the adversary must cut one of the first two rounds of BC and this has to be its first misbehavior in a particular algorithm run.

The number k of rounds in the second phase of the algorithm is chosen randomly by the initiator of the exchange according to a given distribution $(\beta_0, \beta_1, \dots)$ i.e., $\Pr(k = i) = \beta_i$. We assume this distribution to be public. The adversary performs the attack in a given round by dropping a certain subset of messages sent to, or received by, the hosts it controls, i.e., by cutting the channels. When the adversary cuts channels at more than $n/2$ hosts in the same round, we say that it *cuts* the round. Since the adversary does not know in which round BC starts, the best attack consists in choosing a value i according to the distribution $(\beta_0, \beta_1, \dots)$, starting from which adversary cuts all the rounds until the end of the exchange. Cutting messages at less that $n/2$ hosts, or cutting non-consecutive rounds, cannot improve the probability of success of the adversary.

We define the *probability of unfairness* $\Gamma_{(\beta_0,\beta_1,\dots)}$ as the maximum probability that an adversary succeeds, given the distribution $(\beta_0, \beta_1, \dots)$, and the *average complexity* in terms of number of $fake$ rounds as $\Lambda_{(\beta_0,\beta_1,\dots)} = \sum_{i\geq 1} i\beta_i$.

Theorem 5. *Let* $(\beta_0, \beta_1, \dots)$ *denote the probability distribution of the value* k*. The probability of unfairness (for the algorithm of Figure 3) is*

$$\Gamma_{(\beta_0,\beta_1,\dots)} = \max_{i \geq 0}(\beta_i + \beta_{i+1}).$$

Furthermore, we define the probability distribution that we call *bi-uniform*, as well as the *optimal* probability distribution for the algorithm of Figure 3.

Definition 4. *We say that* $(\beta_0, \beta_1, \dots)$ *is a bi-uniform probability distribution of parameter* t *on the interval* $[0, \kappa]$ *if* $\forall i \geq 0,\ \beta_i + \beta_{i+1} = \frac{1}{\lceil \frac{\kappa+1}{2} \rceil}$ *and* $\beta_1 = t$.

Definition 5. *We say that a probability distribution* $(\beta_0, \beta_1, \dots)$ *is optimal (for the algorithm of Figure 3) if there is no other probability distribution* $(\beta'_0, \beta'_1, \dots)$ *such that* $\exists \Gamma > 0,\ \forall i \geq 0,\ \beta_i + \beta_{i+1} \leq \Gamma,\ \beta'_i + \beta'_{i+1} \leq \Gamma$ *and* $\Lambda_{(\beta'_0,\beta'_1,\dots)} < \Lambda_{(\beta_0,\beta_1,\dots)}$.

In other words, a probability distribution $(\beta_0, \beta_1, \dots)$ is optimal if there is no: (1) probability distribution $(\beta'_0, \beta'_1, \dots)$ and (2) probability of unfairness Γ such that the average complexity of $(\beta'_0, \beta'_1, \dots)$, in terms of the number fake rounds, is lower that the average complexity of $(\beta_0, \beta_1, \dots)$.

The following theorem states our optimality result and proves that the probability of unfairness can be made arbitrarily low, by making the algorithm complexity higher.

Theorem 6. *The optimal probability distribution (for the algorithm of Figure 3) is the bi-uniform probability distribution of parameter 0. Moreover, if the distribution is defined on* $[0, \kappa]$ *with* κ *even, the probability of unfairness is* $\Gamma_{bi\text{-}uniform} = \frac{2}{\kappa+2}$ *and the average complexity, in terms of the number of fake rounds, is* $\Lambda_{bi\text{-}uniform} = \frac{\kappa}{2}$.[6]

5 Concluding Remarks

It was shown in [8] that deterministic two-party fair exchange is impossible without a trusted third party. We are not aware of any impossibility result for fair exchange between more than two parties. Published results on such multi-party protocols focus on reducing the necessary trust in the third party [3,10], or focus on contract signing [4,11], a special form of fair exchange. In [12] 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). The use of security modules in fair exchange was already explored in the two-party context: in particular, [23] employs smart cards as security modules to solve two-party fair exchange in an optimistic way, whereas [2] describes a probabilistic solution to two-party fair exchange.

Acknowledgments

We are very grateful to Partha Dutta for his crucial contribution in the impossibility proof of Appendix A, and to Bastian Pochon for his useful comments on the specification of biased consensus.

[6] It can be shown that our GDFE algorithm satisfies $\Gamma_{\text{bi-uniform}}/\Xi_{\text{bi-uniform}} \approx 2/\Lambda_{\text{bi-uniform}}$, where Ξ is the probability that all processes successfully terminate the algorithm; rephrasing the result of [22] in our context would mean that the optimal for a randomized biased consensus is $\Gamma/\Xi \geq 1/\Lambda$ (instead of $2/\Lambda$). We believe that the factor 2 in our case is due to the fact that we ensure biased agreement with a correct majority.

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. F. Bao, R. Deng, K. Q. Nguyen, and V. Varadharajan. Multi-party fair exchange with an off-line trusted neutral party. In *Proceedings of the 10th International Workshop on Database & Expert Systems Applications*, pages 858–862, Florence, Italy, 1–3 Sept. 1999. IEEE Computer Society Press.
4. B. Baum-Waidner and M. Waidner. Round-optimal and abuse-free multi-party contract signing. In *27th International Colloquium on Automata, Languages and Programming (ICALP '2000)*, volume 1853 of *Lecture Notes in Computer Science*, pages 524–535, Geneva, Switzerland, July 2000. Springer-Verlag.
5. P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA, 1987.
6. H. Bürk and A. Pfitzmann. Value exchange systems enabling security and unobservability. *Computers & Security*, 9(8):715–721, 1990.
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. M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, Apr. 1985.
10. M. K. Franklin and G. Tsudik. Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In *Financial Cryptography – FC '98*, volume 1465 of *Lecture Notes in Computer Science*, pages 90–102, Anguilla, British West Indies, Feb. 1998. Springer-Verlag.
11. J. A. Garay and P. MacKenzie. Abuse-free multi-party contract signing. In *Distributed Computing – DISC '99*, volume 1693 of *Lecture Notes in Computer Science*, pages 151–165, Bratislava, Slovak Rep., 27–29 Sept. 1999. Springer-Verlag.
12. 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.
13. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Mateo, CA, 1996.
14. O. Markowitch and Y. Roggeman. Probabilistic non-repudiation without trusted third party. In *Proceedings of the 2nd Workshop on Security in Communication Networks*, 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. P. R. Parvédy and M. Raynal. Optimal early stopping uniform consensus in synchronous systems with process omission failures. Technical Report PI 1509, IRISA, Jan. 2003.
18. K. J. Perry and S. Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, 12(3):477–482, Mar. 1986.
19. M. Raynal. Consensus in synchronous systems: a concise guided tour. In *Proc. Nineth IEEE Pacific Rim Int. Symposium on Dependable Computing (PRDC'02)*, Tsukuba (Japan), Dec. 2002.
20. D. Skeen. Non-blocking commit protocols. In *Proc. ACM SIGMOD Conf.*, page 133, Ann Arbor, MI, Apr.-May 1981.
21. Trusted Computing Group. Trusted computing group homepage. Internet: `https://www.trustedcomputinggroup.org/`, 2003.
22. G. Varghese and N. A. Lynch. A tradeoff between safety and liveness for randomized coordinated attack. *Information and Computation*, 128(1):57–71, 1996.
23. 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.

A Impossibility of Biased Consensus without Correct Majority

We consider the synchronous model of [13], with general omission failure model [18]. Processes proceed in synchronous rounds and may fail by omitting to send or to receive a message, or both.

Note that the properties of *Biased Consensus* (BC) imply that: (a) if some process propose 0, then no process decides 1, and (b) (non-uniform agreement) no two correct processes decide differently. Also by definition, NBAC (non blocking atomic commit) is stronger than BC.

Theorem 2. *No algorithm solves biased consensus if* $\lceil \frac{n}{2} \rceil$ *processes can be faulty.*

Proof. We divide the set of processes into two sets, S_1 and S_2, each containing at most $\lceil \frac{n}{2} \rceil$ processes. In a given run of BC algorithm, the k-round configuration C denotes the configuration of the system at the end of round k in that run. We define some runs which extend a k-round configuration as follows: $run1(C)$ is a run in which, after round k, the processes in S_2 fail in such a way that, in every round after round k and for each $i \in \{1, 2\}$: (1) processes in S_i receive messages from other processes in S_i, and (3) no process in S_i receives any message from any process in S_{3-i}. Basically, the processes in S_2 fail by send omission while sending message to processes in S_1, and fail by receive omission when receiving message from processes in S_1. We define $run2(C)$ symmetrically: a run in which, after round k, the processes in S_1 fail in such a way that, in every round after round k and each $i \in \{1, 2\}$: (1) the processes in S_i receive messages from other processes in S_i, and (3) no process in S_i receives any message from any process in S_{3-i}.

Here processes in S_1 fail by send omission while sending message to processes in S_2, and fail by receive omission when receiving messages from processes in S_2. It is important to observe that no process can distinguish $run1(C)$ from $run2(C)$.

We denote by $val1(C)$ the decision value of processes in S_1 in $run1(C)$. Since all processes in S_1 are correct, they all decide on the same value in $run1(C)$. Since no process in S_1 can distinguish $run1(C)$ from $run2(C)$, the processes in S_1 decide $val1(C)$ in $run2(C)$ as well. Similarly, $val2(C)$ denotes the decision value of processes in S_2 in $run2(C)$, and hence, in $run1(C)$ as well.

Suppose by contradiction that there is an algorithm A that solves BC when $\lceil \frac{n}{2} \rceil$ processes can fail. Consider a run r of A in which every process proposes 1 and no process is faulty. By the *Non Triviality* property of BC, every process decides 1 in r, say at round z. Denote by C_k the configuration of the system at the end of round k in r. We now try to determine $val1(C_k)$ and $val2(C_k)$.

We claim that $val1(C_0)$ is 0. To see why, notice that the processes in S_1 are correct in $run1(C_0)$ and they never receive any message from the processes in S_2. Thus, processes in S_1 cannot distinguish $run1(C_0)$ from $run1(D)$, where D is an initial configuration in which processes in S_1 propose 1 and processes in S_2 propose 0. From *Validity*, processes in S_1 decide 0 in $run1(D)$, and hence, in $run1(C_0)$. Thus $val1(C_0)$ is 0. Similarly, we can show that $val2(C_0)$ is 0.

Clearly, $val1(C_z)$ is 1, because all processes decide (or rather, has already decided) 1 in $run1(C_z)$. Similarly, $val2(C_z)$ is 1.

We now claim that for all rounds i such that $0 \leq i \leq z$, $val1(C_i) = val2(C_i)$. Assume that $val1(C_i) = 0$ and $val2(C_i) = 1$. (The contradiction for $val1(C_i) = 1$ and $val2(C_i) = 0$, is symmetric.) Consider $run1(C_i)$. The processes in S_1 are correct in $run1(C_i)$ and they decide $val1(C_i) = 0$. The processes in S_2 are faulty in $run1(C_i)$, but they cannot distinguish $run1(C_i)$ from $run2(C_i)$ and hence, decide $val2(C_i) = 1$ in $run1(C_i)$. Thus $run1(C_i)$ violates *Biased Agreement.*

Thus, for every round i such that $0 \le i \le z$, $val1(C_i) = val2(C_i)$. Thus there is a round j such that $val1(C_j) = val2(C_j) = 0$ and $val1(C_{j+1}) = val2(C_{j+1}) = 1$. Consider the following $j+1$ round configuration C' that is obtained by extending C_j as follows: processes in S_2 fail (actually commit send omission) such that: (1) for $i \in \{1, 2\}$, the processes in S_i receive messages from other processes in S_i, (2) no process in S_1 receives any message from S_2 in round $j+1$, but (3) the processes in S_2 receive the message from every process in S_1.

Observe that S_2 cannot distinguish C' from C_{j+1} and S_1 cannot distinguish C' from the $(j+1)$-round configuration of $run1(C_j)$. Consider a run r' that extends C' in which S_2 omits to send and receive a message from S_1. Notice that S_1 is correct in r' and S_2 is faulty in r'. However, S_2 cannot distinguish r' from $run2(C_{j+1})$, and hence, decides $val2(C_{j+1}) = 1$ at the end of round z in r'. Furthermore, S_1 cannot distinguish r' from $run1(C_j)$, and hence, decides $val1(C_j) = 0$ at the end of round z in r'. Clearly, r' violates *Biased Agreement*.

B Early Stopping Biased Consensus Algorithm

As we pointed out in the Section 4, our biased consensus algorithm of Figure 4 is an adaptation of the synchronous early stopping uniform consensus algorithm of [17]. Our modifications with respect to the algorithm of [17] is that we have processes agree on a vector of initially proposed values ($\boldsymbol{Vote}$), rather then on the set of those. This is a minor change that does not affect the properties of the algorithm. Basically, every process i stores information about the value initially proposed by some process j at $Vote_i[j]$.

In addition, we introduce *dummy* messages to solve the security requirement of our gracefully degrading fair exchange algorithm and to have the uniform communication pattern. In other words, in our *biased consensus* solution, every process in every round sends exactly one message to every other process, but some (*dummy*) messages are tagged to be disregarded by the receiving process. Our solution assumes that every BC protocol message has the same size.[7]

We now give and prove the properties of our BC algorithm. We do not prove here that all processes that invoke *decide*() function (lines 30-34) agree on the vector $\boldsymbol{Vote}$ (this includes all correct processes if $t < n/2$). Reader interested in this proof should refer to [17]. As discussed above, our algorithm inherently satisfies this property, given $t < n/2$. To summarize, our algorithm satisfies the following properties:

- (Early Stopping) Every correct process decides in at most $min(f+2, t+1)$ rounds.
- (Non Triviality) If no process is faulty or proposes 0, then no correct process decides 0.
- (Validity) No process decides 1 if some process proposes 0.
- (Biased Agreement) If any process decides 1, then no correct process decides 0.

Early Stopping property is inherited from the original algorithm. Consider *Non Triviality*. Assume that all processes are correct and all propose 1. In this case, all processes agree on vector $\boldsymbol{Vote} = 1^n$. Therefore *decide*() returns 1 at every process. Consider now *Biased Agreement* and note that, if any process decides 1 it must have invoked *decide*() and $\boldsymbol{Vote} = 1^n$. This implies that every correct process invokes *decide*(), evaluates the same vector $\boldsymbol{Vote}$ that processes agreed on and, therefore, returns 1. Consider now *Validity* If some process p_j proposed $vote_j = 0$ every process p_i that invokes *decide*() (if any) has $Vote_i[j] = 0$ or $Vote_i[j] = \bot$, as processes agree on

[7] This could be implemented by meeting the maximal size of BC message by padding all message of BC algorithm to reach this size.

```
BCpropose(vote_i) returns decision is

Vote_i := ⊥^n; Vote_i[i] := vote_i; new_i := ⊥^n; new_i[i] := vote_i;
locked_i := ∅; suspected_i := ∅; % r=0 %
for r := 1, 2, ..., t + 1 do % r: round number %
begin_ round
  foreach p_j do
    if p_j ∈ suspected_i then dummy := 1 else dummy := 0 endif
     send (new_i, locked_i, dummy) to G_j
  enddo
  new_i := ⊥^n
   foreach G_j ∉ suspected_i do
     if (new_j, locked_j, dummy = 0) has been received from G_j then
       foreach m ∈ [1 ... n] do
         if (new_j[m] ≠ ⊥) and (Vote_i[m] = ⊥) then
           Vote_i[m] := new_j[m]; new_i[m] := new_j[m]
           locked_i := locked_i ∪ locked_j
         endif
       enddo
     else
       if (G_j ∉ locked_i) then suspected_i := suspected_i ∪ {G_j} endif
     endif
   enddo
   if (|suspected_i| > t) then return (0) endif
   if (G_i ∉ locked_i) then
     if (r > |suspected_i|) or (locked_i ≠ ∅) then locked_i := locked_i ∪ {G_i} endif
   else
     if (|locked_i| > t) then decide(Vote_i) endif
   endif
 end_ round
 decide(Vote_i)
end % of Biased Consensus

Procedure decide(Vote) is
 if (∃m, 1 ≤ m ≤ n, s.t.(Vote[m] = ⊥) or (Vote[m] = 0)) then
   return (0)
 else
   return (1)
 end
end % of decide
```

Fig. 4. Pseudocode of a synchronous, early stopping Biased Consensus algorithm: code of process G_i.

the vector $\boldsymbol{Vote}$ and the coordinate j of $\boldsymbol{Vote}$ is either $\perp$ or $vote_j$. Therefore no process can decide 1. Note that *Validity* holds for any t.

C Correctness of GDFE Algorithm and Optimal Probability Distribution

Theorem 4. *The algorithm of Figure 3 solves gracefully degrading fair exchange.*

The *Timeliness* property is guaranteed by the fact that we consider a synchronous system and the *Termination* property of the underlying BC algorithm.

Consider *Effectiveness* and assume that all participating hosts are honest and all items match their descriptions. All security modules will enter and exit the $fake$ phase having $vote = 1$, so all security modules will *BCpropose* 1. By the *Non Triviality* property of BC every module returns 1 and subsequently returns the expected item to its host.

Now we consider *Fairness*. It is important here to recall that the security modules are tamper-proof and no information leaks from them apart from what is explicitly released through their interface. We first prove a preliminary lemma. For convenience, if the security module returns $\perp$ to its host, we say that security module aborts the GDFE algorithm.

Lemma 1. *If the first round in which some security module G_j aborts the GDFE algorithm is round i ($0 \leq i < k$), then at the end of round $i+1$ every security module has aborted the GDFE algorithm.*

Proof. Because G_j has aborted the GDFE algorithm at the end of round i, no security module will receive G_j's *vote* in round $i+1$. From line 15, it can be seen that every security module will abort the algorithm at latest at the end of round $i+1$ (some modules might have aborted the algorithm in round i, like G_j). □

Consider the case in which the first misbehavior of some of the dishonest hosts occurs in the round i where $0 \leq i < k$ (misbehavior in round 0 includes the initiator's misbehavior or some dishonest host sending the wrong item). According to Lemma 1, by the end of the round $i+1 \leq k$, all security modules will abort the algorithm, so *Fairness* is preserved.

Note that Lemma 1 does not hold for the k-th round. Some dishonest hosts can cut the channels for the first time in that round in such way that some security modules receive all messages and some do not. Hence some modules will *BCpropose* 1 and others will abort the algorithm at the end of round k and will not participate in BC. Because the modules that invoked consensus cannot distinguish this run from the run in which some faulty module proposed 0 and failed immediately in such way that it did not send or receive any message, all security modules that had invoked BC will return 0. At the end, none of the hosts gets the item.

The last possibility is that the first misbehavior occurs during the execution of the BC. This means that every security module has proposed 1 to BC. If there is a majority of honest hosts, the *Biased Agreement* property of BC guarantees *Fairness*. Indeed, *Fairness* can be violated only if some security module returns the expected item to its host, while some correct security module returns $\perp$ to its honest host. From line 18, it is obvious that this would be possible only if some security module returns 1 from BC, while some correct security module returns 0 which contradicts the *Biased Agreement* property.

If the adversary controls half of more of the hosts *Fairness* could be violated. As pointed out in Section 4, this is possible only if the adversary cuts one of the first two rounds of BC. Here, we prove Theorems 5 and 6 and show that even if there is no honest majority, probability of probability of unfairness can be made arbitrarily low and that it is inversely proportional to the average complexity in terms of fake rounds. This completes the proof of Theorem 4. In addition, Theorem 6 gives the optimal probability distribution of the value k.

Theorem 5. *Let $(\beta_0, \beta_1, \dots)$ denote the probability distribution of the value k. The probability of unfairness (for the algorithm of Figure 3) is*

$$\Gamma_{(\beta_0,\beta_1,\dots)} = \max_{i\geq 0}(\beta_i + \beta_{i+1}).$$

Proof. Let γ_i be the probability that the attack succeeds if it starts at round i ($i > 0$). We already know that $\gamma_{i\leq k} = 0$ and that $\gamma_{i>k+2} = 0$. We have therefore:

$$\gamma_1 = \beta_0,\ \gamma_2 = \beta_0 + \beta_1,\ \gamma_3 = \beta_1 + \beta_2, \dots,\ \gamma_i = \beta_{i-2} + \beta_{i-1}, \dots$$

According to the probability distribution $(\beta_0, \beta_1, \dots)$, the maximum probability of unfairness $\Gamma_{(\beta_0,\beta_1,\dots)}$ is therefore

$$\Gamma_{(\beta_0,\beta_1,\dots)} = \max_{i>0}(\gamma_i) = \max_{i\geq 2}(\beta_0, \beta_{i-2} + \beta_{i-1}) = \max_{i\geq 0}(\beta_i + \beta_{i+1}).$$

□

Example 1. If $(\beta_0, \beta_1, \dots, \beta_\kappa)$ is the uniform distribution on the interval $[0, \kappa]$, then $\beta_i = \frac{1}{\kappa+1}$ if $0 \leq i \leq \kappa$ and $\beta_i = 0$ otherwise. We have therefore $\Gamma_{\text{uniform}} = \max_{0\leq i\leq\kappa-1}(\beta_i + \beta_{i+1}) = \frac{2}{\kappa+1}$ and the average complexity in terms of fake rounds is in this case $\Lambda_{\text{uniform}} = \frac{\kappa}{2}$.

Note that Definition 4 of Section 4 implies that, for bi-uniform distribution $\beta_i = \frac{1}{\lceil\frac{\kappa+1}{2}\rceil} - t$ if i is even and $\beta_i = t$ if i is odd. Moreover, t is necessarily equal to 0 if κ is even and $0 \leq t \leq \frac{1}{\lceil\frac{\kappa+1}{2}\rceil}$ if κ is odd.

Theorem 6. *The optimal probability distribution (for the algorithm of Figure 3) is the bi-uniform probability distribution of parameter 0. Moreover, if the distribution is defined on $[0, \kappa]$ with κ even, the probability of unfairness is $\Gamma_{\text{bi-uniform}} = \frac{2}{\kappa+2}$ and the average complexity in terms of number of fake rounds is $\Lambda_{\text{bi-uniform}} = \frac{\kappa}{2}$.*

Proof. First, we prove that if the probability distribution $(\beta_0, \beta_1, \dots)$ is optimal, then it is bi-uniform. In order to prove this assertion, we give Lemma 2.

Lemma 2. *Let $(\beta_0, \beta_1, \dots)$ denote the probability distribution of the value k. Let γ_i be the probability that the attack succeeds if it starts at round i ($i > 0$). We have $\sum_{i\geq 1} \gamma_i = 2$.*

Proof. We have $\gamma_1 = \beta_0,\ \gamma_2 = \beta_0 + \beta_1,\ \gamma_i = \beta_{i-2} + \beta_{i-1},\ \dots$ We have therefore

$$\sum_{i\geq 0} \gamma_i = \beta_0 + \sum_{i\geq 2}(\beta_{i-2} + \beta_{i-1}) = \left(\sum_{i\geq 0} \beta_i\right) + \left(\beta_0 + \sum_{i\geq 1} \beta_i\right) = 2\sum_{i\geq 0} \beta_i = 2.$$

□

Let $\mathit{\Gamma}$ be the probability of unfairness and $\mathit{\Lambda}$ by the average complexity in terms of number of fake rounds; we have

$$\mathit{\Gamma} = \max_{i\geq 0}(\beta_i + \beta_{i+1}) = \max_{i\geq 2}(\gamma_i) \text{ and } \mathit{\Lambda} = \sum_{i\geq 1} i\beta_i.$$

Since $\sum_{i\geq 1}\gamma_i$ is constant, $\sum_{i\geq 2} i\gamma_i$ is obviously minimum when the first γ_is are maximum, that is equal to $\mathit{\Gamma}$. Indeed, suppose that exists $j \geq 0$ such that

$$\gamma_i = \mathit{\Gamma} \text{ if } i < j \text{ and } \gamma_j < \mathit{\Gamma}, \tag{1}$$

then $\exists\epsilon > 0$ such that $\gamma_j = \mathit{\Gamma} - \epsilon$. We have so

$$\sum_{i\geq 2} i\gamma_i = \sum_{\substack{i\geq 2\\ i\neq j}} i\gamma_i + j(\mathit{\Gamma} - \epsilon) + \ell\epsilon$$

where $\ell > j$ (because $\ell \leq j$ contradicts Eq. 1). So $j(\mathit{\Gamma} - \epsilon) + \ell\epsilon > j\mathit{\Gamma}$ implying that if γ_is are not maximum then $\sum_{i\geq 2} i\gamma_i$ is not minimum. Since $\gamma_0 = \beta_0$ and $\forall i \geq 2$ $\gamma_i = \beta_{i-2} + \beta_{i-1} = \mathit{\Gamma}$, we have

$$\begin{aligned}
\sum_{i\geq 1} i\gamma_i &= \beta_0 + \sum_{i\geq 2} i\beta_{i-1} + \sum_{i\geq 2} i\beta_{i-2} \\
&= \beta_0 + \left(\sum_{i\geq 1} i\beta_i + \sum_{i\geq 0}\beta_i - \beta_0\right) + \left(\sum_{i\geq 1} i\beta_i + 2\sum_{i\geq 0}\beta_i\right) \\
&= 2\sum_{i\geq 1} i\beta_i + 3 \\
&= 2\mathit{\Lambda} + 3
\end{aligned}$$

So $\mathit{\Lambda}$ being maximum implies that $\forall i \geq 0$ $\beta_i + \beta_{i+1} = \mathit{\Gamma}$ which further implies that $(\beta_0, \beta_1, \dots)$ is bi-uniform. Note that if $(\beta_0, \beta_1, \dots, \beta_\kappa)$ is a *finite* probability distribution, then $\gamma_i = \beta_{i-2} + \beta_{i-1}$ for $2 \leq i \leq \kappa + 1$, $\gamma_{\kappa+2} = \beta_\kappa$ and $\gamma_i = 0$ if $i > \kappa + 2$.

We prove now that the bi-uniform probability distribution of parameter t is optimal when $t = 0$. As previously, we argue that $\sum_{i\geq 1} i\beta_i$ is minimum when the first β_is are maximum. Since $\forall i \geq 0$ $\beta_i + \beta_{i+1} = \mathit{\Gamma}$, the probability distribution is optimal if $\beta_i = 0$ when i is odd, that is when $t = 0$. Since $\beta_i = 0$ when i is odd, we suppose that κ is even. So, if $(\beta_0, \beta_1, \dots, \beta_\kappa)$ is a bi-uniform probability distribution of parameter 0 such that κ is even, we have

$$\mathit{\Gamma}_{\text{bi-uniform}} = \frac{1}{\lceil\frac{\kappa+1}{2}\rceil} = \frac{2}{\kappa + 2}$$

and

$$\mathit{\Lambda}_{\text{bi-uniform}} = \sum_{i\geq 1} i\beta_i = \sum_{\substack{i\geq 1\\ i \text{ even}}} i\mathit{\Gamma}_{\text{bi-uniform}} = \frac{(\kappa+2)\kappa}{4}\,\mathit{\Gamma}_{\text{bi-uniform}} = \frac{\kappa}{2}.$$

□