Secure Multiparty Computation vs. Fair Exchange - Bridging the Gap

Benoît Garbinato, Ian Rickebusch

Secure Multiparty Computation vs. Fair Exchange — Bridging the Gap (Invited Paper)

Benoît Garbinato[1], Ian Rickebusch[2]

[1] Université de Lausanne, Lausanne, Switzerland
benoit.garbinato@unil.ch

[2] ELCA Informatique, Lausanne, Switzerland
ian.rickebusch@elca.ch

Abstract. In this paper, we propose a comparison of *fair exchange* and *secure multiparty computation*. Despite their apparent similarity, these two problems arise respectively from the fields of distributed systems and of modern cryptography. The wide differences of description and approach in these research fields render hazardous a straightforward comparison of the various results and solutions. By introducing a common specification framework for the two problems, we examine the differences regarding their generality and properties, which then leads us to specify a new problem, named the *generalized fair computation*. This third problem helps understanding the relations between secure multiparty computation and fair exchange, by bridging the gap between them. We conclude with a discussion on the possible origins of the confusion surrounding certain results.

1 Introduction

The problem of *secure multiparty computation* (SMC) comes from the field of modern cryptography and consists in allowing a group of parties to compute a specific function securely, i.e., with the input of each party remaining private to all others. The motivation behind this problem is to study to what extent a group of mutually distrustful parties can emulate a centralized *trusted third party* (TTP). Accordingly, the SMC problem is set in a *real* model, with no TTP, and is specified with respect to an *ideal* model, in which all the computation is executed by a centralized TTP. The problem of *fair exchange* (FE) on the other hand comes from the field of distributed systems and consists in exchanging values between mutually distrustful parties. The exchange is fair, if either all parties obtain their desired values or no party obtains anything useful.

This paper proposes a comparison of these two problems, which are confusingly similar. On the one hand, they share a close connection with the concept of TTP, thus appearing somehow equivalent. On the other hand, several results found in their respective literature appear to be contradictory. Because the specifications of both problems differ greatly in form, making sense out of these

confounding results is not straightforward. Moreover one can find very little in the literature about measuring one problem against the other.

Secure Multiparty Computation. The SMC problem was first introduced by [33] and has been extensively studied since then. The definition of the behavioral constraints was first presented through descriptions of properties [33] and later by relying on the *simulation* approach [10, 20, 22], in which the level of security in the real model is defined by emulation of an ideal model relying on a trusted third party (TTP).

Whether described by a set of properties or by the simulation approach, the level of security has been of much debate in the literature [22]. Yet it now seems to be standard that the focus is on providing *privacy* and *correctness* but not *termination* or *fairness* [22, 28], i.e., *secure computation with abort*. Indeed, based on [25] the authors of [22] argue that, since Byzantine agreement cannot be composed with two thirds or more Byzantine processes, *security* should be decoupled from *agreement*, which is closely related to *termination* and *fairness*. Based on this last argument, the definition of SMC that we consider in this paper corresponds to *secure computation with abort*.

Not surprisingly, research on SMC has also produced an impressive body of work regarding solutions, impossibilities and lower bounds. In [7, 11, 30] for instance, it is sequentially shown that any multiparty protocol can be achieved in an unconditionally secure manner, provided that the system is synchronous and that there is an honest majority of peers. These results even provide some level of fairness, as also does [21]. In the hierarchy of [22] mentioned above, the level of fairness in [30] matches the second definition, also known as *fair computation*.

Fair Exchange. The fair exchange problem comes basically in two flavors, namely a *weak* variant and a *true* variant [31]. 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 [26]. Among these properties, *fairness* is the most difficult to capture and hence where most specifications tend to differ, as in [1, 4, 29]. 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. Many researches also explicitly aim at the fair exchange variant involving only two peers [1, 2, 15, 27, 32], 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, and consists in fine-grained properties, each one capturing either a liveness or a safety requirement [18].

Some authors also discuss the difficulty of fair exchange and propose impossibility results in various models. In [29], fair exchange is measured against consensus, and an impossibility result on fair exchange in asynchronous models is shown by comparison with the FLP impossibility [14]. In [13], fair exchange

is shown to be impossible to solve *deterministically* in an asynchronous system with no TTP. 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* [16].

Most solutions to fair exchange rely on a TTP 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 [9]. A TTP brings synchronism and control over terms of the exchange 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 [1, 6, 9, 27]. However *optimistic* approaches are based on the strong assumption that the environment is mostly honest. By relying on fully decentralized tamperproof modules, in [17], our approach departs from the traditional TTP-based approach and bears similarities with [3–5], which also assume embedded tamperproof modules. However in order to ensure true fair exchange, in [18] we introduce a connectivity condition, named the reachable majority condition, which states that all correct processes must be connected reliably to a majority of trustees.

Roadmap. Section 2 first introduces an integrated specification framework based on specifications found in both modern cryptography and distributed systems. This section then proposes to apply this new specification framework to describe SMC and FE, in order to allow a first comparison. Section 3 introduces a new problem, named *generalized fair computation* (GFC), that bears the same level of generality as secure multiparty computation and the same behavioral constraints as fair exchange. Section 4 discusses some aspects that we believe are at the origin of the confusion between fair exchange and secure multiparty computation, while Section 5 summarizes our contribution.

2 Problem Comparison

We consider a distributed system consisting of a set Π of n fully interconnected *processes*, $\Pi = \{p_1, \ldots, p_n\}$, which 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. The *execution* of algorithm A is then defined as a sequence of steps executed by processes of Π. In each step, a process has the opportunity of atomically performing all of the following actions: (1) send a message, (2) receive a message and (3) update its local state. In each step, the process can of course choose to skip any of these actions. Based on this definition, a *Byzantine process* is one that deviates from A in any way, so a Byzantine process is Byzantine against a specific algorithm A. Byzantine failures can indeed only be defined with respect to some algorithm [12].

2.1 An Integrated Specification Framework

The fields of modern cryptography and distributed systems apply their own distinct approaches when describing the problems of SMC and FE, respectively.

However both problems are comparable to that of a cooperative multi-player game. The specifications are divided into two distinct parts: (1) a *functional definition*, which is equivalent to the goal of the game, and (2) *behavioral constraints*, corresponding to the rules under which the game is played.

Functional Definition. Modern cryptography describes the functional definition using a mathematical function, whereas distributed systems rely on a set of primitives. While these approaches may appear different, they are quite similar, with the former being slightly more formal. In modern cryptography, the functionality is described as a whole, with all the inputs and outputs, whereas in distributed systems, it is usually described using one primitive to allow each process to provide its input and a second primitive to allow each process to receive its output value. For example, modern cryptography would define a function f: $(y_1, \ldots, y_i, \ldots, y_n) = f(x_1, \ldots, x_i, \ldots, x_n)$. In distributed systems, this function would be translated into the two following primitives along with a description of their semantics:

input(x_i) – Enables process p_i to provide its input x_i.
output(y_i) – Allows process p_i to obtain output y_i. (Works as a callback.)

For the functional definition, the use of a mathematical function is probably preferable, since it is more precise than the description of primitives.

Behavioral Constraints. Modern cryptography usually defines the constraints regarding acceptable and unacceptable behaviors by analogy with a model relying on a *trusted third party* (TTP), named the *ideal* model [20, 22]. Thus the constraints are described by saying that the *real* model, i.e., one not relying on a TTP, should ensure the same properties as in the *ideal* model. In contrast, the field of distributed systems relies on an exhaustive set of properties describing both what should happen and what cannot happen, named respectively liveness and safety properties [23]. Commonly found properties include, e.g., *termination*, which is a liveness property stating that all processes eventually obtain an output value, and *uniqueness*, which is a safety property stating that no process can obtain more than one output value. If the simplicity of the analogy with the *ideal* model seems appealing, we argue that it is inadequate and possibly misleading. This is indeed the case because the guaranties offered by a TTP are too strong and are thus impossible to emulate in the *real* model. The only possible way for the *real* model to live up to the *ideal* model is to downgrade the *ideal* model to a *not-so-ideal* model [20]. Thus, although relying on a set of properties may be less intuitive, it is nonetheless a preferable approach, especially when comparing SMC and FE.

2.2 Secure Multiparty Computation

The *secure multiparty computation* (SMC) problem consists in a group of processes trying to securely compute a common function with the inputs from all processes. The difficulty of the problem resides in achieving *privacy*, i.e., keeping each input hidden from all other processes. Note that Byzantine processes may omit to provide an input value, in which case the computation may not terminate.

Functional Definition. More formally, each process is required to compute securely the result of a well-known deterministic function: $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, where x_i and y_i are respectively the inputs and outputs of process p_i. Interestingly, in the SMC literature, the domain of definition is usually not clearly defined. This is probably because it is not really an issue when dealing with *correctness* and *privacy*.

Behavioral Constraints. Secure multiparty computation allows processes to securely compute the result of a specific function, with each process providing an input value and expecting to receive an output value as the result of the computation. In order to achieve this cooperative goal, there are a number of behavioral constraints to respect, which are described by the set of properties hereafter. At the end of the computation of the function f, we say that process p_i *outputs* a value, meaning that the function returns the output value to p_i. This convention comes from the field of distributed systems and is similar to the one used for classical *deliver* primitives, e.g., in reliable broadcast primitives [24].

The SMC problem is usually defined in the modern cryptographic literature by relying on a TTP [20], since the problem was originally described to provide the same guarantees as a TTP. However, in the perspective of bridging the gap between SMC and FE, we define the semantics of SMC using a set of abstract properties. We aim at matching as closely as possible the definition given in [20], which corresponds to the commonly accepted definition of SMC in the field of modern cryptography.

Validity. If a correct process p_i outputs a value y_i, then y_i was computed using function f and with at least the inputs of all correct processes.[3]

Uniqueness. No correct process outputs more than one value.

Non-triviality. If all processes are correct, every process outputs a value.

Privacy. No process p_j outputs the input value x_i or output value y_i of any correct process p_i, apart from what is possibly given away by inputs and outputs of Byzantine processes.

The *validity* and *privacy* properties are intrinsic to the specification of the SMC problem, while *uniqueness* implicitly derives from the ideal model based on a TTP. The *non-triviality* property can be found in most – possibly in all – distributed systems problems, as it becomes evidently necessary when trying to provide any meaningful solution. According to [20], ensuring a liveness property, e.g., *termination*, is not part of the specification of the SMC problem. This is not the case in [8], where the authors rely on secure modules and are thus able to ensure *termination*. However, we argue that [8] describes a stronger problem than the usual problem of SMC and thus, accordingly, omitting *termination* from our specification better captures the essence of SMC as commonly defined in modern cryptography.

2.3 Fair Exchange (FE)

The fair exchange problem consists in a group of processes trying to exchange inputs specified ex ante, i.e., each process provides a specific input and expects the input of some other process in exchange. The difficulty of the problem resides

[3] In the literature of SMC, *validity* is usually referred to as *correctness*.

in achieving *fairness*. Intuitively, fairness means that, if one process obtains its desired output, then all processes involved in the exchange should also obtain their desired output.

Functional Definition. More formally, each process is required to compute the result of a deterministic function F: $(y_1, \ldots, y_n) = F(x_1, \ldots, x_n)$, where x_i and y_i are respectively the inputs and outputs of process p_i. When all processes are correct, the outcome of F is defined by a function f, a permutation with no fixed points, with input domain $X = X_1 \times \ldots \times X_n$, where $X_i = \{\overline{x}_i\}$, a set containing a single specific value, and output domain $Y = Y_1 \times \ldots \times Y_n$, where $Y_i = \{\overline{y}_i\}$ and $(\overline{y}_1, \ldots, \overline{y}_n) = f(\overline{x}_1, \ldots, \overline{x}_n)$. Thus the terms of the exchange are defined by f, X and Y. Function f therefore provides the outputs of F when the inputs provided are those expected, i.e., they belong to the domain of f, and the computation achieves completion.[4] However, if this is not the case, F outputs a special value φ for all processes. This special value φ indicates that the exchange aborted. Function F is defined as follows:

$$F(x_1, \ldots, x_n) = \begin{cases} f(x_1, \ldots, x_n), & \text{if } \forall x_i,\ x_i \in X_i \text{ and } f \text{ achieves completion;} \\ \forall y_i, y_i = \varphi, & \text{otherwise.} \end{cases}$$

Behavioral Constraints. Fair exchange allows processes to exchange input values in a fair manner. Each process inputs a value in exchange for a counterpart, of which it can verify the validity. The computation is concluded when every process outputs either a value y_i, which is by definition the result of function f, or the abort value φ, meaning that the computation aborted. The behavioral constraints of the problem of fair exchange are described by the set of properties hereafter.

Validity. If any correct process p_i outputs value y_i, then either $y_i \in Y_i$ or $y_i = \varphi$.
Uniqueness. No correct process outputs more than one value.
Non-triviality. If all processes are correct, no process outputs the abort value φ.
Termination. Every correct process *eventually* outputs a value.
Privacy. No process p_j outputs the input value x_i or the output value y_i of any correct process p_i, apart from what is possibly given away by inputs and outputs of Byzantine processes.[5]
Fairness. If any process p_i outputs a value y_i, with $y_i \in Y_i$, then every correct process p_j outputs a value y_j, with $y_j \in Y_j$, unless p_i is Byzantine and y_i is computable from the inputs of Byzantine processes.

Of these six properties, the last two, *privacy* and *fairness*, are specific to the problem of fair exchange and define precisely the possible outcomes of fair exchange algorithms. The *privacy* property ensures that no process outputs the input or output values of a correct process. This does not prevent a Byzantine process from illicitly outputting the input or output of some other Byzantine process, since such a behavior cannot be prevented and does not prejudice any correct process.

Our set of six fine-grained properties describing fair exchange was first introduced in a companion paper for clarity reasons [16], to capture the problem

[4] Byzantine behaviors may prevent the completion of the computation of f.
[5] Corresponds to the *Integrity* property in [16–18].

better and to facilitate reasoning about the different results and solutions. Other specifications of fair exchange usually rely on a single property to capture the notion of *fairness* [1, 4, 29]. 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 [18]. The choice of two properties (*privacy* and *fairness*), instead of just one, happens to be particularly relevant when comparing SMC and FE, since SMC ensures only *privacy*.

2.4 Comparing SMC and FE

With both problems described using our integrated specification framework, differences that were somehow hidden by implicit assumptions or merely because of the heterogeneous specifications, now become apparent. In this section, we take a first look at the gap between SMC and FE. The differences between the two problems obviously depend on the choices we made regarding what to include in their definitions, in particular for SMC, whose exact definition is still up for debate. Nonetheless, the comparison provides an interesting clarification of the relationship between SMC and FE.

The first major difference is found in the generality of the function computed in both problems. In SMC, the specification allows any function f to be considered, whereas fair exchange only deals with permutations with no fixed points and specific constraints on the inputs of all processes, as described by the terms of the exchange. Secure multiparty computation is far more general than fair exchange with respect to the range of functions it considers. In other words, they are from different levels of abstraction.

The second explicit difference lies in the lists of properties. Fair exchange ensures two properties, *termination* and *fairness*, not ensured by secure multiparty computation. While these properties can be found in some particular specifications of SMC, they do not belong to the specification commonly used in the modern cryptography literature. Indeed, from [20], we learn that a peer cannot be prevented from abruptly interrupting any execution of SMC at any time, which denies the possibility of ensuring *termination*. Since ensuring true fairness obviously relies on the termination of any execution, this aspect also impacts the possibility of ensuring *fairness*. In fact, *fairness* is usually not considered in SMC, even though this might seem the case from the emulation approach of a TTP-based ideal model. So not only is *fairness* not ensured, it is also seldom discussed, hence the confusion when comparing seemingly conflicting results in SMC and FE. Moreover, when *fairness* is indeed mentioned in the context of SMC, it usually has a weaker connotation. This last aspect is further discussed in the Section 4.

To summarize, SMC achieves more in terms of generality, while FE provides two additional properties, i.e., *termination* and *fairness*. This leads us to introduce a third problem, which provides both the generality of secure multiparty computation and the set of properties of fair exchange.

3 Bridging the Gap

In order to bridge the gap between *secure multiparty computation* (SMC) and *fair exchange* (FE), we introduce a third problem named *generalized fair computation* (GFC). SMC better relates to GFC than to FE, since both SMC and

GFC encompass the same level of generality. Moreover FE connects to GFC as one of its instances, so it belongs to a family of problems all deriving from GFC.

3.1 Generalized Fair Computation

The *generalized fair computation* (GFC) problem consists in a group of processes trying to compute the result of specific function in a fair manner. Intuitively, fair means that, if one process obtains its desired output, then all processes involved in the computation should also obtain their desired output. In a companion paper [19], we provide a fully decentralized solution to GFC, relying on tamperproof security modules.

Functional Definition. More formally, each process is required to compute the result of a deterministic function F: $(y_1, \ldots, y_n) = F(x_1, \ldots, x_n)$, where x_i and y_i are respectively the inputs and outputs of process p_i. When all processes are correct, the outcome of F is defined by a function f with input domain $X = X_1 \times \ldots \times X_n$ and output domain $Y = Y_1 \times \ldots \times Y_n$: $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$. Function f thus provides the outputs of F when the inputs are those expected, i.e., they belong to the domain of f, and when the computation achieves completion. However, if such is not the case, F outputs a special value φ for all processes. This special value φ indicates that the computation aborted. Function F is thus defined as follows:

$$F(x_1, \ldots, x_n) = \begin{cases} f(x_1, \ldots, x_n), \text{ if } \forall x_i,\ x_i \in X_i \text{ and } f \text{ achieves completion;} \\ \forall y_i,\ y_i = \varphi, \quad \text{otherwise.} \end{cases}$$

Behavioral Constraints. Generalized fair computation allows processes to compute, fairly and securely, the result of a specific function, with each process providing an input value and expecting to receive an output value as the result of the computation. The computation is completed when every process outputs either a value y_i, which is by definition the result of function f, or the abort value φ, meaning that the computation aborted. The behavioral constraints of GFC are identical to those of fair exchange. However, we now see that adding *termination* and *fairness* to the list of properties of SMC has an influence on the *validity* property. Indeed, *termination* implies having a special abort value in cases where executions do not proceed as expected. This impacts the *validity* property, which needs to be modified to account for the special outcomes.

Furthermore, the functional definition also has an impact on the *validity* property, since it requires that all inputs be verified. Accordingly, in the *validity* property, the correct outcome y_i is computed with the inputs of all processes, not just with at least the inputs of correct ones. However, in the case of certain functions that may be correctly computed in the absence of inputs from Byzantine processes, the input verification would arbitrarily restrict the correct outcomes. A simple countermeasure to this drawback is achieved by introducing an omission value $\perp$ in each element X_i of the domain X. The value $\perp$ is then used in the computation of f in place of missing inputs of Byzantine processes.

3.2 Solving SMC and FE using GFC

By construction, GFC is a generalization of FE and they both share the same set of properties. Thus, building FE on top of GFC simply consists in defining

function f as a permutation, with no fixed points, and in limiting the input and output domains, X and Y, to n singletons matching the terms of the exchange. More interestingly, a solution to SMC can be built on top of GFC, showing that the problem of GFC is indeed stronger than the the problem of SMC.

Algorithm 1 SMC executed by participant p_i

```
1: Uses: Fair Secure Multiparty Computation (GFC)
2: function input(x_i)
3:     GFC.input(x_i)                          {calls the input primitive of GFC}

4: upon GFC.output(y'_i) do                    {callback from GFC}
5:     if y'_i ≠ φ then                        {Checks if value is not the abort value φ}
6:         output(y'_i)                        {outputs the result of GFC}
```

Algorithm 1 provides a solution to SMC, with function f, relying on a specific GFC module with function $f' = f$. The algorithm is fairly straightforward as it first calls the input primitive of the GFC module and then, if that value is not the abort value φ, it outputs the value provided by the GFC module. The correctness proof of Algorithm 1 is also fairly obvious since GFC ensures exactly the same *uniqueness*, *non-triviality* and *privacy* properties as SMC. Thus it only remains to show that Algorithm 1 ensures the *validity* property of SMC.

Theorem 1 (Validity). *If a correct process p_i outputs a value y_i, then y_i was computed using function f and with at least the inputs of all correct processes.*

Proof. If the SMC module of a correct process p_i outputs a value y_i at line 6, y_i is any value y'_i output by the GFC module at line 4, excluding the abort value φ. From the validity property of GFC, either $y'_i \in Y'$ or $y'_i = \varphi$. From the definition of GFC, when $y'_i \neq \varphi$, y'_i is computed using f' and with the inputs of all processes. So, since $y_i \neq \varphi$ and $f' = f$, y_i was computed using function f and with at least the inputs of all correct processes.

4 Discussion

From their respective literature, it is understandably difficult to distinguish precisely one from the other, since they have much in common but are described using different vocabularies. The main reason is probably to be found in the close connection of both with the notion of *trusted third party* (TTP). However, a TTP ensures a large set of security properties but the focus of each problem is on a different subset of these properties. Hereafter, we propose a discussion, hopefully a clarification, on the possible origins of confusion: (1) the seemingly contradictory results and (2) the TTP emulation approach taken in SMC.

Contradictory results? On the one hand, when assuming an honest majority, SMC can be made unconditionally secure, even in the context of *secure computation with unanimous abort and complete fairness* [30], i.e., fair computation. On the other hand, fair exchange is impossible to solve without at least one identified process that all processes can trust [16]. So either one of these results

is wrong or they are simply not addressing the same issue. Indeed, if these two statements seem contradictory, it is because of their common context and the lack of accuracy found in natural language, i.e., the notion implied by the word *fairness* is different from one case to the other.

In the case of fair exchange, in order to respect the terms of the exchange, the notion of fairness implies a constraint on the *inputs* of all processes, including those of Byzantine processes. There is no such constraint in the case of *fair computation*, since the validity of inputs of Byzantine processes is usually not much of a concern. The concern is to ensure that adversaries may not obtain information from the inputs of correct processes, i.e., either in absolute (*privacy*) or with respect to the information correct processes are able to obtain (*fairness*). In other words, what makes fair exchange more difficult to solve than fair computation is the constraint on the inputs of all processes. The verification of all the inputs is a key step in solving fair exchange in Byzantine environments and needs to be done by some trusted process, be it a TTP or trustees. This decisive difference in the notion of *fairness* also explains why the domain of the function describing the SMC problem is usually not explicitly defined, whereas it is a crucial element in fair exchange.

An ideal model? The simulation approach was introduced in SMC in order to simplify the description of the behavioral constraints. The aim is to provide the same level of security as a TTP but without a TTP, i.e., in a fully decentralized setting. However, results from both fair exchange and secure multiparty computation have shown that achieving the same level of security as a TTP is simply impossible. In order to make the emulation possible, certain properties ensured by the TTP have to be dropped. For example, in the ideal model, malicious processes are assumed to be able to interrupt the computation of the TTP, e.g., by cutting off all the communications with the TTP. While this assumption is made because ensuring that the computation is not interrupted is indeed impossible in the real model, it is nonetheless contradictory with the very notion of TTP.

In the light of the above restrictions on this alleged ideal model, one can argue that introducing such an approach in order to avoid having to produce an exhaustive list of properties *ensured* by SMC, e.g., *correctness* and *privacy*, is slightly less convincing. First because the impossibility of unconditionally emulating a TTP forces us to provide an exhaustive list of properties that should be *ignored* in the TTP, e.g., *fairness* or *termination*. And secondly because it is somewhat misleading to readers, who are not necessarily aware of the implicit *not-so-ideal* model hidden behind the *ideal* model. Furthermore, since the ability to interrupt the TTP in the ideal model is not so important for *privacy* or *correctness*, this restriction is not always explicitly stated in SMC papers.

5 Conclusion

In this paper, we proposed to bridge the gap between SMC and FE by introducing an integrated specification framework, borrowing elements from both the fields of modern cryptography and distributed systems. Using this, we translated the description of the two problems, which allowed us to point out a certain number of differences that were obscured by the heterogenous specifications. This led

us to introduce the problem of *generalized fair computation*, which combines the generality of secure multiparty computation and the behavioral constraints of fair exchange. Finally, in the light of what precedes, we concluded by discussing the possible origins of the confusion between them.

References

1. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Area in Communications*, 18:593–610, 2000.
2. G. Ateniese. Efficient verifiable encryption (and fair exchange) of digital signatures. In *CCS '99: Proceedings of the 6th ACM conference on Computer and communications security*, pages 138–146, New York, NY, USA, 1999. ACM Press.
3. G. Avoine, F. Gärtner, R. Guerraoui, K. Kursawe, S. Vaudenay, and M. Vukolic. Reducing fair exchange to atomic commit. Technical report, Swiss Federal Institute of Technology (EPFL), 2004.
4. G. Avoine, F. Gärtner, R. Guerraoui, and M. Vukolic. Gracefully degrading fair exchange with security modules (extended abstract). In *In Proceedings of the 5th European Dependable Computing Conference - EDCC 2005*, 2005.
5. G. Avoine and S. Vaudenay. Fair exchange with guardian angels. In *Proceedings of the 4th International Workshop on Information Security Applications (WISA)*, volume LNCS. Springer, 2003.
6. B. Baum-Waidner and M. Waidner. Round-optimal and abuse free optimistic multi-party contract signing. In *Automata, Languages and Programming*, number 1853 in Lecture Notes in Computer Science (LNCS), pages 524–535. Springer, 2000.
7. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1988. ACM Press.
8. Z. Benenson, F. Gartner, and D. Kesdogan. Secure multi-party computation with security modules. Technical report, RWTH Aachen University of Technology, 2004.
9. H. Bürk and A. Pfitzmann. Value exchange systems enabling security and unobservability. *Computers & Security*, 9(9):715–721, 1990.
10. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(1):143–202, 2000.
11. D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *STOC '88: Proceedings of the 20th ACM symposium on Theory of computing*, pages 11–19, New York, NY, USA, 1988. ACM Press.
12. A. Doudou, B. Garbinato, and R. Guerraoui. *Dependable Computing Systems: Paradigms, Performance Issues, and Applications*, chapter Tolerating Arbitrary Failures with State Machine Replication, pages 27–56. Wiley, 2005.
13. S. Even and Y. Yacobi. Relations among public key signature systems. Technical report, Technion - Israel Institute of Technology, 1980.
14. M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*, 32:374–382, April 1985.
15. M.K. Franklin and M.K. Reiter. Fair exchange with a semi-trusted third party (extended abstract). In *CCS '97: Proceedings of the 4th ACM conference on Computer and communications security*, pages 1–5, New York, NY, USA, 1997. ACM Press.
16. B. Garbinato and I. Rickebusch. Impossibility results on fair exchange. In *Proceedings of the 6th International Workshop on Innovative Internet Community Systems (I2CS'06)*, volume LNCS. Springer, 2006.

17. B. Garbinato and I. Rickebusch. Orchestrating fair exchanges between mutually distrustful web services. In *Proceedings of the ACM Workshop on Secure Web Services (SWS'06)*. ACM Press, 2006.
18. B. Garbinato and I. Rickebusch. A topological condition for solving fair exchange in byzantine environments. In *Proceedings of the 8th International Conference on Information and Communications Security (ICICS'06)*, volume LNCS. Springer, 2006.
19. B. Garbinato and I. Rickebusch. Secure multiparty computation vs. fair exchange: Bridging the gap. Technical Report DOP-20070123, University of Lausanne, DOP Lab, 2007. http://www.hec.unil.ch/dop/Download/articles/DOP-20070123.pdf.
20. O. Goldreich. *The Foundations of Cryptography*, volume 2. Cambridge University Press, 2004.
21. S. Goldwasser and L.A. Levin. Fair computation of general functions in presence of immoral majority. In *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, pages 77–93, London, UK, 1990. Springer-Verlag.
22. S. Goldwasser and Y. Lindell. Secure computation without agreement. In *DISC '02: Proceedings of the 16th International Conference on Distributed Computing*, pages 17–32, London, UK, 2002. Springer-Verlag.
23. R. Guerraoui and L. Rodrigues. *Introduction to Reliable Distributed Programming*. Springer-Verlag, Berlin, Germany, 2006.
24. V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. pages 97–145, 1993.
25. Y. Lindell, A. Lysyanskaya, and T. Rabin. On the composition of authenticated byzantine agreement. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 514–523, New York, NY, USA, 2002. ACM Press.
26. O. Markowitch, D. Gollmann, and S. Kremer. 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*, pages 451–464. Springer, November 2002.
27. S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 12–19, New York, NY, USA, 2003. ACM Press.
28. S. Micali and P. Rogaway. Secure computation (abstract). In *CRYPTO*, pages 392–404, 1991.
29. H. Pagnia and F. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Swiss Federal Institute of Technology (EPFL), 1999.
30. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85, New York, NY, USA, 1989. ACM Press.
31. I. Ray and I. Ray. Fair exchange in e-commerce. *SIGecom Exchanges*, 3(2), 2002.
32. I. Ray, I. Ray, and N. Natarajan. An anonymous and failure resilient fair-exchange e-commerce protocol. *Decision Support Systems*, 39(3):267–292, 2005.
33. A.C. Yao. Protocols for secure computation. In *23th Symposium on Foundations of Computer Science (FOCS)*, pages 160–164. IEEE Computer Society Press, 1982.