Fair Exchange

Henning Pagnia[1], Holger Vogt[2] and Felix C. Gärtner[3]

[1]*University of Cooperative Education, Mannheim, Germany*
[2]*Department of Computer Science, Darmstadt University of Technology, Germany*
[3]*École Polytechnique Fédérale de Lausanne, Switzerland*
Email: fcg@acm.org

The growing importance of electronic commerce and the increasing number of applications in this area has lead research into studying methods of how to perform safe and secure online business transactions over the Internet. A central problem in this context is that of *fair exchange*, i.e., how to exchange two electronic items in a fair manner. We give a general introduction into the research area of fair exchange and discuss several formalizations of fairness. We find that, although a considerable number of fair exchange protocols exist, they usually have been defined for special scenarios and thus only work under particular assumptions. Furthermore, these protocols provide different degrees of fairness and cause different communication overhead. To alleviate this, we present a generalizing framework defining a suite of protocol modules which allows us to implement different fair exchange protocols. Depending on the properties of the exchanged items an appropriate fair exchange protocol can be selected and applied. Our study is accompanied by a comprehensive survey of the relevant literature.

Received month date, year; revised month date, year

1. INTRODUCTION

"Electronic commerce" (short: *e-commerce*) via the Internet is one of the most important markets today and is expected to flourish for at least the next decade. While there is still a notable fraction of companies that utilize the Internet solely for advertising and company-internal communication, some have started to use the network for the full range of business transactions including the sale of goods and services. This is particularly attractive in the case of *digital* services which can be entirely rendered via an electronic network. Examples for such services are the delivery of video or audio data, the electronic purchase of computer software, the transfer of digital money, the writing of a digital receipt, or querying a database, but also the provision of telephone lines or Internet access.

Several business models have been adopted, for example subscriptions are in common use. Here, the customers must subscribe to the service and pay a fixed amount of money on a regular basis, e.g., monthly, for using the service. Alternatively, pay-per-use can serve as a different business model having the advantage that customers do not need to tie down to a specific service provider and potentially even allowing them to keep their anonymity.

The advantages of pay-per-use are so evident that we expect the broad establishment of pay-per-use applications for digital services in the near future. Protocols supporting this must be carefully constructed because they have to take special properties of the diverse services into consideration. A common property of many digital services is that they normally cannot be revoked, i.e., once the service has been granted the service provider has no effective means to force the recipient to return it without keeping a private copy. This is particularly crucial if the two business partners reside in different countries where different regional law regulations apply. Therefore, the exchange of the digital services (with the payment being one of them) should at best be performed simultaneously in order to guarantee fairness for both involved parties. Unfortunately, real simultaneousness cannot be achieved because in general, granting a digital service comprises the transmission of several bits so service delivery is not atomic. This also implies that delivery always requires a certain amount of time and, hence, during the exchange a network failure might occur or either party might intentionally interrupt the transmission at any time. The exchange process might end unfair if in such a situation one party has already completed service delivery while the other has not.

As an example consider two mutually anonymous parties wanting to simultaneously disclose their identity. For this, they can exchange digital identity cards which are electronically signed by a trusted issuer in order to prevent unauthorized modification. In order to ensure fairness (i.e., simultaneousness) a third party must be involved which may at first collect both identity cards and forwards them subsequently. Since both business partners must trust the third party to perform

correctly and honestly, the third party is referenced to as a *trustee*. In order to reduce the trustee's load different protocols have been proposed. A special class of *optimistic protocols* [1] incorporates the trustee only in case of failures or conflicts. Others weaken the degree of fairness which is achieved.

An additional problem occurring in this context is how each party can check if the other has correctly delivered the service and whether it is the desired one. In our example, each party could betray by sending not its identity card but something else (e.g., today's weather forecast). If the checking for correctness lies in the responsibility of each party then fairness cannot be guaranteed since a honest party unveils its identity while a malicious party does not. With an active trustee as described above the problem can easily be solved by letting the trustee perform the necessary correctness checks (i.e., verifying whether the exchanged items are valid identity cards). However, still a party might deliver somebody else's valid identity card instead of its own. In order to protect against this fraud, a *public key infrastructure* and a *challenge-response protocol* can be utilized. The example demonstrates that correctness checks play an important role to fair exchange and that protocols intended to solve the fair exchange problem should not leave them unconsidered.

The fair exchange protocols which have been presented in literature are diverse and largely incomparable even in the amount of fairness they offer. We will show that many existing protocols (including optimistic ones) can be understood as a composition of different protocol modules with distinct functionality. By separating the concerns and identifying these modules we are able to construct a general framework for modeling fair exchange protocols and thus give a better understanding of the fair exchange process itself. We thereby focus on the general interaction pattern of these protocols rather than the cryptographic primitives used to implement them.

The remainder of this article is structured as follows. We state our system assumptions, give some informal definitions of fair exchange, and discuss special properties of what is exchanged in Section 2. In the same section, we examine how items can be specified and how their validity can be checked. Subsequently, in Section 3 we describe several approaches to rigorously formalize the fair exchange problem and discuss under which assumptions it can be solved. In Section 4, we introduce a generalizing framework which allows us to implement several different fair exchange protocols. Then we can select the appropriate protocol depending on the properties of the exchanged items and on the required fairness level. After providing an extensive review of the work relevant to the context of fair exchange in Section 5, we conclude our paper with a summary and a discussion of our approach in Section 6.

2. DEFINING FAIR EXCHANGE

In this section we will informally define the major terms used in this paper. Many of these terms will be formalized in Section 3.

2.1. System Assumptions

A fair exchange takes place within a network of computers which are connected e.g., through the Internet. We model this situation as a distributed system consisting of a set of processes which can send and receive messages between each other over reliable FIFO communication channels. This means that mechanisms exist which ensure that no messages are lost or altered in transit and that they are eventually delivered to the destination process. (Under certain reasonable assumptions reliable channels can be implemented on top of unreliable channels [2].) The delays introduced by these mechanisms among other reasons justify the assumption that the system is *asynchronous* [3] meaning that there is no bound on the message delivery delay or on the relative processing speeds of the processes.

A *problem specification* is a high-level description of the required behavior of the processes involved. Informally spoken, a specification states under which conditions certain process events must never occur and under which conditions other events must eventually occur. A *protocol* is a low-level description of the actual system behavior. It is a precise description about which actions a process should perform in response to actions of other processes. A particular execution of a protocol satisfies a specification if the execution adheres to the event descriptions of the specification. Due to the nondeterminism of the network a protocol may have several different executions. A protocol satisfies (or implements) a specification if all its executions satisfy the specification.

Fair exchange protocols usually involve the use of symmetric and asymmetric crypto systems [4] allowing to encrypt, decrypt or digitally sign data. We assume that the communication channels are secured through appropriate means that ensure integrity, confidentiality and authenticity of information while in transit and that protocol messages contain unique identifiers and sequence numbers to prevent replay of messages or parts of them.

We model malevolent parties involved in the fair exchange by processes which do not follow the protocol they are supposed to follow, i.e., they might arbitrarily stop executing steps or send corrupted messages. However, it is assumed that malevolent parties are not capable to encrypt, decrypt or digitally sign messages unless they have access to the necessary keys. Non-malevolent parties do not crash and behave according to their protocol.

2.2. Informal Definitions

We now give an intuitive understanding of the fair exchange problem. We follow the presentation of Asokan [5] with modifications made by Pagnia and Vogt [6]. We assume two participating parties A and B. Each party starts with an electronic item i and a description d of what that party would like to receive in exchange for i. In the notation we will identify the items using subscripts, i.e., i_A is A's item and d_B is a description of B's desired item.[2] We assume that there exists a verification function which takes an item and a description and returns the value *true* only if the item matches the description. There are two possible termination states of the protocol, either *success* or *abort*. Both parties have means to check whether and in which state the protocol has terminated.

A *fair exchange protocol* is a protocol which implements the following three requirements:

1. (Effectiveness) If both parties behave according to the protocol, both parties do not want to abandon the exchange and both items match the description then, when the protocol has completed, A has i_B and B has i_A and both reach a *success* termination state.
2. (Termination) A party which behaves according to the protocol will eventually reach either a *success* or an *abort* termination state.
3. (Fairness, preliminary definition) If at least one party does not behave according to the protocol or if at least one item does not match the description, then no honest participant wins or loses anything valuable.

The formulation of the Effectiveness requirement is rather delicate since it is only required to hold "if both parties do not want to abandon the exchange" [7]. This precondition is necessary because in asynchronous systems messages can be delayed for an arbitrary (but finite) amount of time. Hence, at any time it is not possible, say for a well-behaving party A, to distinguish the two cases where either (a) party B is correctly following the protocol but the network is slow, or (b) B has stopped to follow the protocol overall [8]. Hence, the precondition allows to define a very weak notion of a timeout in asynchronous systems with the risk that exchanges do not succeed because parties "time out" too early.

The same reasons (i.e., asynchrony) may lead to a different notion of termination, which we call *weak termination*. Briefly spoken, weak termination means that the final state of the exchange of, say, party A depends on an action performed by party B. Effectively, this means that if B does *not* perform this action, A will never be notified. Hence, A will never "know" the final outcome of the exchange. For example, consider the case where A has started an exchange with B and is waiting for a message from B upon which the protocol outcome for A depends. As long as A is still waiting, both outcomes (success or abort) are still possible. If A *never* receives this message, the outcome should be abort. But due to asynchrony, A cannot decide whether a message from B will arrive or B has stopped to participate in the exchange. Due to this fact, to make the correct termination decision, A would have to wait forever, which is undesirable. Note that A still has more guarantees than "no termination": For example, if A by default decides on abort and changes to success in case a message is received, A can still rely on the fact that *eventually* its decision will be correct. With "no termination", even this is not guaranteed. Obviously, this weakened version of termination should be avoided if possible. In general, the protocols described in this paper achieve the stronger notion of termination. If not, we will discuss the reasons why this may be sufficient.

Definitions of fair exchange often differ in their understanding of the fairness requirement. For example, Asokan [5] distinguishes between strong and weak fairness. The fairness requirement described above is called strong fairness. For weak fairness it is required that — in case of a failed exchange — either party can prove that it has behaved correctly, i.e., it has followed the prescribed exchange protocol. As part of an external dispute, the proof must be shown to an external arbiter, e.g., a court of justice, who has the power to establish fairness, usually by forcing both parties into cooperation. The problem with this is that in most countries it is still unclear what such a proof might be in order to be acceptable as evidence in a lawsuit.

In any case, a lawsuit is expensive and its outcome might be rather uncertain. Therefore, it is desirable to resolve as many conflicts as possible within the exchange system itself. If the arbiter is sufficiently powerful it can automatically process the proofs and decide how to proceed. The advantage of this is that conflicts are now automatically processed *within* the exchange system, thus increasing the degree of fairness.

In extension to the definitions of Asokan [5], we propose the following hierarchy of fairness guarantees [9]:

F_6: Fairness can be guaranteed automatically by the system without further communication with the other party.
F_5: Fairness can be guaranteed automatically by the system with eventual cooperation of the other party.
F_4: Fairness can be achieved automatically by the system through providing a compensation for a suffered disadvantage.
F_3: Fairness can only be guaranteed outside of the system by an external dispute without further cooperation of the other party.
F_2: Fairness can only be guaranteed outside of the system by an external dispute with eventual coopera-

[2] Note that Asokan [5] uses d_A to denote the description of the item desired by B.

tion of the other party.

F_1: Fairness can only be achieved outside of the system by an external dispute by providing a compensation for a suffered disadvantage.

F_0: No fairness.

The fairness definitions F_6 to F_4 are supposed to be stronger than the others because conflicts can be resolved automatically without the need for a subsequent external dispute. In general, we would like to achieve a level of fairness which is as high as possible in the hierarchy. However, higher levels of fairness are more difficult to ensure.

Strong fairness by the definition of Asokan [5] corresponds to F_6. As Asokan does not make any assumptions about the willingness of the parties to cooperate, F_5 can also be regarded as strong fairness in the sense of Asokan [5]. Gärtner et al. [10] call F_5 *eventually strong fairness*. The difference between F_6 and F_5 lies in the additional assumption made about the participants, i.e., they can be eventually forced to cooperate. For example, if a participating party stores all its data on a remote computer which is managed by an independent service provider, it may be possible to technically enforce access to this data by "official" means (i.e., through a special message signed by the arbiter). In general, fairness F_5 can be achieved by using a trusted computing environment [11] on the user's machine. The term "eventual cooperation" still makes sense in asynchronous systems if the help of such official means is not called for immediately.

The categories F_3 and F_2 match Asokan's weak fairness definition. The distinction between F_3 and F_2 is similar to that made between F_6 and F_5 only that means outside of the technical fair exchange system are referenced. For example, consider the case where a misbehaving participant is brought in front of court and is sentenced to pay for a received item. If the participant now reluctantly pays the debts, then fairness F_2 is achieved since the assumption of "basic cooperation" is made about a misbehaving party. However, even if this assumption is not made, it is possible to collect debts in practice even without cooperation of the other party. For example, calling a bailiff (or the sheriff) ensures fairness F_3.

The categories F_4 and F_1 describe a different fairness concept in which it is assumed that a non-delivered item can be substituted by a different one (e.g., a payment) which compensates the loss. Because this does not match the original intention of the exchange process we have ranked compensation as a method to achieve fairness weaker than the others.

2.3. Properties of Exchanged Items

Certain item properties can be exploited by fair exchange protocols. Some of these properties refer to the item descriptions which participating parties must give before engaging in fair exchange. Before discussing these properties and ways on how to ensure that items in fact have these properties in Section 2.4, we will first consider a different class of properties, namely those that can help the third party to resolve conflicts. In this section, we describe some of these properties in more detail, mainly *generatability* and *revocability* [5].

2.3.1. Generatability

A generatable item is an item which can be generated by the trustee in case the receiving party can prove that it has behaved correctly. Depending on the effectiveness of generatability we distinguish:

Strong generatability: It is guaranteed that the trustee will be able to generate such an item.

Weak generatability: The trustee can try to generate such an item, but he may fail in generating it, if a participating party has misbehaved (e.g. by sending a garbled message). If this is the case, the trustee will always detect such a misbehavior and he will be able to provably determine the cheating party.

The difference between strong and weak generatability is that the strong notion always ensures successful item generation whereas weak generatability can only succeed, if nobody has misbehaved. Nevertheless weak generatability can still support conflict resolution, if just one party misbehaved: As fairness can only be provided to honest parties (i.e., parties which follow the specified protocol), the trustee is allowed to disadvantage a provably misbehaving party in order to restore fairness for the honest party.

We now give some examples to illustrate our definition of strong and weak generatability. These examples show different methods to make arbitrary items generatable. In the first two examples strong generatability is achieved, while the last two examples result in weak generatability.

1. A party forwards a copy of its item to the trustee who checks it against the item description and stores it for a possible subsequent dispute. The party is provided with a signed receipt which can be presented to any other party as a proof that the item is generatable by the trustee.
2. A party forwards a copy of its item to the trustee who checks it according to a given item description. Then the trustee encrypts the item and signs the encrypted item together with the description. Finally the party is provided with the signature, the encrypted item and the decryption key. During an exchange the party can use the encrypted item and the description together with the signature as a proof that the trustee is able to generate the item by decrypting the encrypted item.
   The advantage of this approach compared to the first one is that the trustee only needs to store the

decryption key for a possible dispute. This significantly reduces the storage space required at the trustee.

3. A party encrypts its item with a random key. This key is then deposited at the trustee who returns a receipt for it, which the party can from now on use as a proof for the weak generatability of the item. Note, that this receipt is not a proof for the strong generatability of the item, since it cannot be guaranteed that the encrypted item can successfully be decrypted with the key which is stored at the trustee.
4. A party encrypts its item with a random key, encrypts this random key with the trustee's public key and signs both ciphertexts as a commitment that the trustee will be able to decrypt these values. The ciphertexts and the signature are then forwarded to the other party as a proof for the generatability of the item. The receiving party can neither decrypt the random key nor the item, but the trustee can do so if the encryption has been performed correctly. This results in weak generatability, because — if the item generation failed — the trustee can at least detect who is responsible.

These examples manifest a tradeoff between the strength of generatability and the burden placed on the trustee: The burden on the trustee decreases from method 1 to method 4 whereas the generatability property is weakened. While in method 1 the trustee must store the entire item, he only needs to store the decryption key in method 2 and 3. Method 4 is the most efficient one in terms of storage space and communication: the trustee can use a single key — namely the own private key — for decrypting any item which was made generatable and the trustee is only contacted in the case of a failure.

It is also possible to utilize some specific item properties in order to make items generatable. Especially in order to make digital signatures strongly generatable certain techniques have been developed which are referred to as *verifiable escrow* [12, 13]: A signature can be encrypted with the public key of the trustee and sent to the other party together with a cryptographic proof that really a correct signature was encrypted. Then the receiver knows that the trustee can decrypt (and thus generate) the signature. Often this approach is also called "verifiable encryption" [12, 14–17]. A different, quite efficient implementation of verifiable encryption relies on off-line coupons [13]. These coupons are retrieved from the trustee in advance and simplify the verification that the correct signature was encrypted. Another approach for verifiable escrow [18, 19] is based on convertible signatures [20]. Here an undeniable signature, which is only interactively verifiable, can be converted to an ordinary signature by a trustee, if the customer asks the trustee to generate this signature.

Special item properties also help to generate signatures on a contract [7]: The trustee may be granted the right to issue a replacement for a signed contract. If this replacement is later equally accepted as a valid contract, the trustee can generate a missing signature on a contract by simply generating this replacement.

There are other items with special properties which are relevant to generatability, for example payments based on electronic coins (e.g., ECash [21]). Basically, such a payment consists of a signature from the issuing bank and thus it can be made generatable as described for digital signatures using verifiable escrow. However, this may not be enough to achieve strong generatability: In an on-line payment system like ECash a payment is only valid, if it contains a valid signature, which has not been deposited in another payment before. This means that even though the signature is strongly generatable, the generation of the payment might fail, if it has been deposited earlier. Then weak generatability can only be achieved, if the bank can tell the trustee whether the customer or the merchant tried to cheat by spending the coins twice.

2.3.2. Revocability

An item is called revocable if a trustee can revoke it in case it has sufficient evidence to do so. Thus, revocation can be used to undo an exchange that cannot be completed. We distinguish two different forms of revocability:

Strong revocability: It is guaranteed that the trustee will be able to revoke such an item.

Weak revocability: The trustee can try to revoke such an item, but he may fail in revoking it. However, such a failure proves that the weakly revocable item really has been delivered to the party that desired this item.[3]

It should be noted that items must not be revocable by a party itself. Otherwise, after a correctly terminated fair exchange, one of the parties could easily revoke the delivered item and thus gain an unfair advantage over the other party. The consequence is that the exchange of such items can never be regarded as fair.

The following examples illustrate the difference between strong and weak revocability.

1. Several payment systems (e.g., credit cards) support revocation of payments. To achieve strong revocability a payment system should only permit revocation, if a trustee requests to do so.
2. In practice many payments will only be revocable for a certain time period (e.g., for some weeks). After that the revocation will fail, which means that only weak revocability is achieved, if it is not

[3] This latter property is sometimes called non-repudiation of receipt [22].

guaranteed that revocation will always be started in time.

3. Digital certificates may grant its owner certain rights: Travel agencies may sell their customers digital tickets that grant them a seat on a certain flight. If only the trustee (maybe in cooperation with the airline) can revoke such a ticket, we achieve weak revocability, because revocation will fail after the customer has taken this flight. But then the trustee has a proof that the customer has to pay for this flight.
   This example also applies to other scenarios, where a customer buys a right to use some service (e.g., a cinema ticket, access to a database, etc.). However, it is essential that only the trustee can revoke such a right of a customer.

2.3.3. Compensation

Compensation means that a trustee will be able to provide a participant with compensation instead of the expected item. It is a necessary requirement that both parties estimate the value of the expected item and the compensation to be equal. Then the exchange will still be fair, if compensation is delivered instead of the expected item.

Compensation can be viewed as a special form of generatability, as the trustee is able to generate something of equal value. However, there is one big difference to generatability: In case of compensation there is the danger that the participant has received the item, but additionally asks for compensation. This may be unfair, if the participant can benefit from utilizing both, the item and the compensation. In contrast, this problem does not exist for generatable items, as their delivery is *idempotent* meaning that it makes no difference whether the item is received once or multiple times.

2.3.4. Time-Sensitivity

In practice, items which lose their value over time exist. For example, a birthday present should be delivered till the birthday party starts, the TV program for Monday is useless on Tuesday, and only the most recent stock market information will be useful. According to Asokan [5] such items are called *time-sensitive*.

The difficulty with time-sensitive items is that nobody but the receiver knows when a sent item really arrived. Thus the receiver may claim that there was some delay until he got the item and that it was already useless then. But the receiver is not able to convince somebody else that the item arrived too late — no matter if this is true or not. Even if a trustee is involved, he will not be able to decide correctly whether there was some delay that made the item useless or not. If the delivering party has to resend an updated version of the time-sensitive item, then this may be unfair, if the receiver is able to utilize both items. Again we see that the exchange of items, whose transmission is not idempotent, can lead to some problems. Dealing with time sensitive items in asynchronous systems may seem impossible because due to arbitrary message delays there can be no guarantee that time-sensitive items are still useable when they arrive. However, dealing with such items may still be possible. The idea is to separate receiving the item over the network and delivering it to the user by utilizing trusted hardware on the user's machine [23, 24]. Before delivery, the user is asked whether the item still has value to him. If yes, the item is delivered immediately, as only local communication without significant delay is required. Otherwise the item is discarded. As the use of secure and tamper-proof hardware is a research topic of its own, we will not consider time-sensitive items in the rest of this paper.

2.4. Item Validation

Apart from "technical" properties (like *generatability* which enables conflict resolution), exchanged items must also satisfy the item description given by the parties at the beginning of the exchange. Specifying these properties as part of the description and validating the items against it is an important but often neglected problem which is inherent to all exchange protocols.

Validation means to check that the items are "as expected" and is the basis for any effective business transaction, even non-electronic ones. In order to be able to do this it is important that a sufficiently detailed description exists for both items. For some kind of items, for example digital money or digital signatures, the validation is rather simple. Another example is a widely used software package which can be checked by computing a cryptographic hash value and comparing it against a trusted reference value which is publicly available [25]. Problems with this solution however can occur if the software contains a serial number or an individual watermark for copyright protection. Other items too are difficult to check: For example, a common description of a software package usually contains a list of features which cannot be checked during a formal verification process. Promises like "high performance" are not precise enough in order to be verified formally and easily cause overexpectation. Therefore, for an accurate validation a complete and formal description is necessary. Unfortunately in many cases this will be impossible to derive, in most other cases it will be very costly. However, we assume that for the items a sufficiently accurate description exists against which they can be verified.

A straightforward solution to the fair exchange problem is based on the concept of an *active trustee*. If such a trustee is available, the exchange can proceed as follows: Both parties, A and B, send their item to the trustee. The trustee checks whether A has sent the item which B wants to have and vice versa. If both checks succeed, the trustee forwards A's item to B and B's item to A. This type of protocol is a *fair exchange protocol with an active trustee*.

The validity check of the items must be performed by the trustee in order to ensure fairness. But note that because the items in consideration for exchange will usually be different ones in each instance of the problem, the activity of checking the items will be different every time the trustee is used. Thus, prior to executing the exchange, both A and B must indicate to the trustee how he should check the other's item for validity.

In the context of mobile code, one solution is for both A and B to devise a specific check method which takes the other's item as an input and returns `true` only if the item actually has the desired features. This approach is rather flexible because it enables to check an item in all ways allowed by the underlying programming language. For example, it is possible to calculate cryptographic checksums over the entire item or over parts of it, it could compare parts of the item with some test data, or it could perform file format checks (e.g., "is the data a JPEG image?").

Subsequently, A devises a method `checkB` and B devises a method `checkA`. Both A and B send the code of this methods to the trustee before the fair exchange takes place. Within the fair exchange protocol, the trustee uses `checkB` to check the item sent by B and `checkA` to check the item sent by A. Only if both methods return `true` the trustee will complete the exchange.

The problem with this solution stems from its flexibility: If arbitrary code is allowed within the check routines, it is possible to cheat. To understand this, imagine that A devises a routine `checkB(item)` which first sends `item` to itself (i.e., to A) and then returns `false`. As a result, the exchange is in danger of ending in an unfair situation where A has obtained B's item but B has not received A's item. Consequently, it is necessary to guarantee that nothing "bad" happens within the check routines.

The "bad things" which may happen refer to the presence of information flow from inside the check routine to another party. This should include information flow through hidden channels [26], which is particularly difficult. In some contexts, it is even necessary to have information flow from inside to outside of the check routine. For example, if one item consists of electronic money, it is impossible to prevent double spending without online access to a bank. This means that the check routine *must* make an online query when checking the money. In this case we can only ensure that the amount of information flow out of the check routine is bounded.

It is sometimes useful for a party to run the other party's check routine on an item before engaging in the fair exchange to prevent selling an item which is inappropriate in advance. However, a problem exists if a party wants to perform "semantic" checking by using heuristics. Instances of this problem are, e.g., plausibility checks on a credit card number or simple spot checks on mass-produced articles. Consider for example the following scenario: A is looking for a text containing a set of keywords and so it devises a method `checkB` which simply scans the item for these words. The code of this method is sent to B so that B can validate that nothing bad happens therein. But knowing the code of `checkB`, B can easily fool A by piecing together an arbitrary text containing these keywords. While not being unfair in a formal sense, the disclosure of its customized check routine lowers a party's confidence in the quality of the received item.

Next we describe three possible solutions for the problem of item checking. Thereby we make it possible to select the best solution for a specific scenario.

2.4.1. Parameterized check routines

One solution is to use a predefined check routine from a specialized library. For example, the trustee could provide a set of publicly accessible methods, e.g., to check whether a file represents a valid JPEG image. In this case either party could simply send the identifier of the desired check routine to the trustee.

Alternatively, the trustee could offer a generic check routine which is parameterized with a predefined range of values. This could be, for example, a routine which scans the input for a set of keywords (the parameter of the routine would be the set of words). Extending this idea, the generic check routine would take an expression in a formal "item description" language as an argument. A drawback of this approach is that this requires the definition of a rather complex language which is sufficiently expressive in order to allow the check for all relevant item properties.

2.4.2. Syntactic Check

Instead of using generic check routines, A and B could still be allowed to write their own methods. In this case it must be possible to automatically verify that their respective routines play according to the rules. For example, by automatically scanning the parties' code before invoking it, it is possible to check whether a method contains the invocation of a `send` command. If no other way exists to smuggle information out of the check function, then fairness can be guaranteed. However, for leaking information usually other possibilities than using the `send` command exist. Fraudulent parties are likely to garble their harmful actions within innocent looking code, and preventing the parties from performing such malicious actions solely by using syntactic checks is a difficult task.

2.4.3. Sandboxing

A more powerful solution for verifying that the check routine well-behaves is to monitor its execution. Any attempt to execute an unfair command will be recognized at run-time. It will result in an immediate termination and rejection of the check routine. This approach for protecting the participating parties from malicious code is comparable with the sandbox model for Java applets. Compared to the syntactic check, sandboxing is

more immune against innocent looking, obfuscated code which is intended to deceive the other party. Another advantage of this security concept is that the timely termination of the check routines can also be monitored. If the execution time exceeds a previously defined time limit, the trustee stops computation and aborts the exchange. A limitation of sandboxing is its flexibility: Any command must either be allowed or forbidden. If for example communication is forbidden, then it is impossible to obtain certificates from a key server.

The solutions again manifest a complex tradeoff between (1) the level of confidence attainable, (2) the flexibility of describing desired item properties, and (3) the complexity of "verifying" the check routines. Parameterized check routines and a syntactic check are probably the methods most easily implementable in practice. However, they either lack flexibility in describing item properties (parameterized check routines) or confidence in the "safety" of the check routine (syntactic check). Sandboxing on the other hand offers both these advantages but requires complex mechanisms and programming language support.

2.5. Summary

This section has introduced the fair exchange problem as the combination of effectiveness, termination and fairness properties. From the literature we know that fairness can be understood in many different ways and we have proposed a hierarchy of well-separable fairness definitions to alleviate this problem. The efficiency and correctness of fair exchange protocols critically depends on specific properties of the exchanged items. We have discussed the most important of these properties, mainly the strong and weak notion of generatability and revocability, and have given examples to manifest their importance. Finally, we have studied the problem of item validation and presented ways on how to increase the security of this important task. All this builds the foundation for concrete protocol design: We show in Section 4 how to exploit the discussed item properties to build fair exchange protocols which achieve varying levels of fairness.

3. FORMALIZING THE FAIR EXCHANGE PROBLEM

The more practically motivated definitions of fairness ("nobody wins or loses something valuable") which have been presented in section 2.2 are precise enough for a general understanding of the concept. However, if it comes to a more rigorous verification of fair exchange protocols, the present definitions lack the necessary level of detail to be of use. In this section we survey the existing attempts to completely formalize fairness in the context of electronic commerce to make fair exchange protocols amendable to rigorous verification. This section is aimed at more theoretically interested readers and can be skipped if practical interests are of more concern.

3.1. Formalizations based on game theory

Game theory can be used to rigorously define the fairness condition of exchange protocols. In this approach, the individual moves of the participants within a protocol are turns in a multi-player game. Briefly spoken, a protocol is fair if any player cannot achieve an advantage over the other player without the other player gaining a similar advantage during the game.

Buttyán [27] uses this approach and completely formalizes fairness using the notions of game trees and strategies [28]. A *strategy* is a description of how a player moves in response to another player's move and a *game* is a pair of strategies. Specific fair exchange protocols can then be formalized as a game (s_A*, s_B*) for two correctly behaving players A and B. A misbehaving player, say A, is modeled as a player who follows a different strategy s_A (involving moves that were not part of the original protocol). The advantages which different players can achieve at different points in the game must be estimated and formalized as a payoff function. A positive value indicates an advantage (i.e., gaining access to the other player's item), a negative value a disadvantage. A protocol is said to be *fair* if for every possible strategy s_A of A, the outcome of the game when A plays s_A and B plays s_B* (i.e., B behaves correctly) is such that A can have a positive payoff only if B has a positive payoff as well. Of course the same must hold vice versa.

3.2. Cryptographic definitions of fairness

Modern cryptographic methods allow to precisely define the security of a system. For example, in the *simulateability paradigm* [29, 30] a relation called "at least as secure as" is established between two systems S_1 and S_2, where S_2 usually is an *ideal* system and S_1 a *real* system supposed to implement the same service as the ideal system. Briefly spoken, S_1 is *at least as secure as* S_2 iff the "view" of the honest user in S_1 is indistinguishable from its view in S_2. Indistinguishable means that the two views are either identical or that the probability that any probabilistic polynomial-time "distinguisher" algorithm can compute the difference is negligible [30, p. 109]. Note here that statements about security are always probabilistic and are based on explicit restrictions on the power of an adversary (i.e., polynomially bounded computation resources).

While not based on game theory, Asokan, Shoup and Waidner [14] also use the term "game" to define the fairness property of such an ideal exchange system. In this game, where participating parties are polynomially bounded interactive turing machines, a well-behaving party A and the trustee T follow their protocol in a purely reactive fashion. The execution of the game is

driven by an adversary $B*$ which has complete power over the network and whose actions are merely restricted by the following rules:

- $B*$ cannot sign or decrypt messages for which $B*$ does not have the appropriate key, but can interact arbitrarily with T obtaining T's signature on adaptively chosen messages.
- Whenever A signals its intention to make a move or wants to receive a message, then $B*$ eventually lets A proceed in that move and eventually supplies the next message to A, respectively.
- Interactions of A with T are not restricted by $B*$.

The restrictions imply that A can follow its protocol and eventually terminate by outputting an item e_A. In this case, $B*$ also terminates and outputs e_{B*}. $B*$ wins the game if $e_{B*} = i_A$ but $e_A \neq i_{B*}$. A protocol is fair if the probability that $B*$ wins the game is negligible.

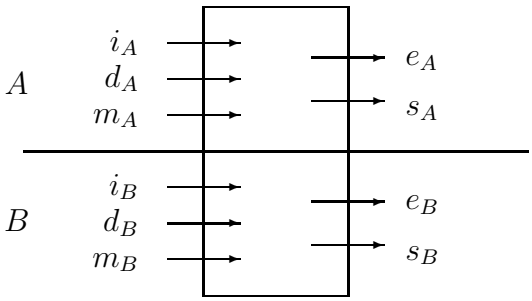
Overall, rigorously proving that a protocol in fact satisfies the requirements of fair exchange remains nontrivial. This is especially true for the above definitions of fairness or if the system model is randomized. In these cases, machine support using verification tools (like model checkers) is practically impossible so proofs must be done by hand and remain a cumbersome, error-prone affair.

3.3. Trace-based formalizations of fairness

There have been attempts to formalize fairness using trace-based concepts from concurrency theory for which mechanical proof support systems exist. An example is the formalism of Cervesato *et al.* [31] in which Chadha, Kanovich and Scedrov [32] have studied fair contract signing protocols. In this section, which is based on Gärtner *et al.* [10], we study the basic possibilities of formalizing fairness in this context.

3.3.1. A fair exchange system

Within this theory, a system is considered to be a black box within an environment. The system interacts with the environment through a set of input/output variables called the *interface*. While the environment and the system may read input and output variables, the former may only be written by the environment while the latter may only be written by the system. A fair exchange system can be modeled as in Fig. 1, i.e., as a black box with three input variables i, d, m and two output variables e and s per party. Again, we will subscript these variables with the identifier of the party to which they belong, e.g., i_A is the input item of party A. The variables i and d have the same meaning as explained in Section 2.2, i.e., i is the input item and d is the description of the desired item. The variable s indicates the state of the protocol and can evaluate to *success* (in case of a successful protocol completion), *abort* (unsuccessful protocol completion) or an undefined value $\perp$ (protocol is still running). In case of a successful protocol completion, e should contain the exchanged item. The variable m is a specification variable indicating whether or not a party is malevolent, i.e., the party will not behave according to the protocol.



i_x input item
e_x output (exchanged) item
d_x description of desired item
m_x flag indicating malevolence
s_x success/abort indication

FIGURE 1. A fair exchange system.

3.3.2. States, traces and properties

A *state* of a system like the one shown in Fig. 1 is an assignment of values to the variables of the interface. A *trace* (or *execution*) is a (possibly infinite) sequence of states of the system and a *property* is defined as a set of traces. A system in itself defines a property, i.e., the set of all traces which the protocol running within the black box may produce at the interface. A system Σ *satisfies* a property P iff all traces of Σ are in P.

3.3.3. Safety and liveness

There are two special kinds of properties called *safety* and *liveness*. A *safety property* informally states that "something bad will never happen". Safety properties can be thought of as invariants, i.e., sets of states which usually describe the "safe", "legal" or "good" states of the system. Formally, a safety property S is a set of executions which is prefix-closed, i.e., for every execution $e \in S$ and every prefix α of e, α must also be in S. Partial correctness and the effectiveness condition of fair exchange are examples of safety properties.

A *liveness property* informally states that "something good will eventually happen". A finite trace α is said to be *live* for a property L, if there exists an execution $e \in L$ of which α is a prefix. A property L is called a *liveness property* if every partial execution is live for L. Intuitively, a liveness property states that any partial system execution α is "not lost", i.e., it is always possible to extend α to still reach some goal. Termination, starvation freedom, and eventual delivery are examples of liveness properties.

3.3.4. Strong fairness as a safety property
Safety and liveness were introduced by Lamport [33] and later formally refined by Alpern and Schneider [34]. It has been argued that almost all important system properties can be expressed as a combination of a safety and a liveness property [34, 35]. Consequently, there have been attempts to model the fairness requirement of fair exchange within this context.

In a previous paper [10], the present authors have investigated the question whether fairness is a safety or a liveness property. The category F_6 of strong fairness formulated as

- (Strong Fairness) If B behaves correctly and if the protocol has terminated, it is never the case that A has B's item and B does not have A's item; and vice versa.

was shown to be a safety property. The inherent assumption behind this formalization is that items are atomic (i.e., they can only be exchanged in their entirety) and not revoked after protocol termination. If we assume that the output can only be written once, the property is a safety property because we can tell in finite time whether it is violated (i.e., after both outputs have been written). Using a safety property is a natural way of formalizing fairness because it is close to invariant-based protocol analysis which is rather common [36]. While other authors often do not separate their correctness conditions into safety and liveness parts, the relevant aspect of fairness is usually a safety property [32, 36].

3.3.5. Fairness as a liveness property
In the same direction, the notion of weak fairness was investigated. Informally, weak fairness means that a possible disadvantage can occur but that it can be eventually refuted. If the disadvantage within the system is not permanent, i.e., there exist means within the system (e.g., by requesting help from a trusted third party) to eventually refute the disadvantage, then this notion of weak fairness can be formalized as a liveness property. Since the outcome of the exchange is similar to strong fairness, this form was called *eventually strong fairness* and corresponds to fairness category F_5. To refute a disadvantage, there must exist some additional assumptions about a misbehaving party. Such an assumption could be, for example, that a misbehaving party will (or can be forced to) eventually cooperate. (Note that the means to force a party to cooperate are fully within the automated system.) However, such assumptions are not very realistic in the usual settings of electronic commerce.

Schneider [37] investigates system properties which can be automatically enforced onto a system by mechanical means. He presents a theoretical result which shows that liveness properties cannot be automatically enforced without making additional assumptions about the misbehaving node. Thus, specifying weak fairness as a liveness property has important practical implications. If we cannot guarantee a certain behavior of the participating nodes, weak fairness as "liveness" cannot be achieved within the automated system.

3.3.6. Weak fairness as a safety property
However, if the disadvantage were permanent (i.e., there are no means to refute it within the system), a protocol must ensure that a participating party has collected enough evidence to prove to an arbiter outside of the system that it behaved correctly. In this direction, fairness would be regarded as a safety property again. Gärtner *et al.* [10] argue that this is *weak fairness* in the sense of Asokan [5] which corresponds to F_2/F_3.

3.4. Impossibility results

3.4.1. Fair exchange without a trusted third party
Recall the situation of fair exchange where two parties A and B want to exchange atomic digital items over a message passing infrastructure. At some point in time, the items must be transmitted over the network. But who should send the item first in a situation of mutual distrust? No matter who begins, there is always the danger of the protocol ending in an unfair situation.

In 1980, Even and Yacobi [38] elaborated this idea to formally prove that there is no fair exchange protocol without a trusted third party in these situations. The proof is by contradiction: Assume a protocol exists which achieves strong fairness in the given scenario without the help of a trusted intermediate. The protocol must consist of a sequence of communication rounds. A communication round consists of sending a message from A to B and back again. Because the protocol is a fair exchange protocol, it must terminate. More specifically, at some point in time A must receive B's item, i.e., after n communication rounds A has sufficient information about B's item but this is not true after $n-1$ communication rounds for A. Because of mutual distrust, B will obviously only participate in round n if it has acquired sufficient information about A's item in some round $n' < n$. However, this contradicts the fact that the protocol used achieves strong fairness. Even and Yacobi conclude that there is no fair exchange protocol under these assumptions. Work by Schunter [30] has investigated similar impossibilities for optimistic contract signing protocols.

3.4.2. Properties of a trusted third party
Since fair exchange is impossible without a TTP, it is natural to investigate the minimal system conditions under which fair exchange is *possible*. What properties must a third party satisfy in order to be helpful in the exchange? While a formal investigation into the precise meanings of these terms is still lacking, two aspects seem particularly relevant in this context: *trustworthiness* and *availability*.

Trustworthiness means that the parties which partic-

ipate in the exchange "know" the protocol which the TTP is executing. This ensures that a TTP will not conspire with one of the participating parties. Knowledge about the TTP's (expected) behavior is only achievable by detailed code inspection and the use of secured hardware. This task is usually delegated to experts and so trusting the TTP means trusting these experts. Certain protocols can slightly reduce the level of trust in the TTP. For example, in some protocols [25, 39] the TTP alone cannot gain useful information about the exchanged items. Other protocols [7] make certain forms of misbehavior of the TTP detectable by the participants.

Availability means that it is not only important to know which protocol the TTP is executing, but also *that* the TTP will follow the protocol, i.e., that the services of the TTP will eventually be available. This means that the TTP must satisfy certain reliability requirements. In asynchronous systems, we conjecture that the minimal requirements are that the TTP must have stable storage and be at least *eventually-up* in the *crash-recovery* model of distributed systems [40]. This means that the TTP may store information persistently and will eventually respond to every request by one of the participating parties.

3.4.3. Separation of item validation and exchange

The discussion about reliability features of the trusted third party has implicitly focused on its exchange functionality. To satisfy fairness, it is not only important to exchange items "atomically" but also to validate the items at some point during the protocol. Validation can be partially delegated to the participants: The idea of this approach is to encrypt the items with random keys and have the participants check certain properties on the encrypted items [41]. However, in this approach the item validation problem recurs when trying to exchange the keys themselves.

Exchange and validation of items needs not be done by the same trustee. It is technically possible to provide two separate trusted hosts: one which performs the exchange and one which is tailored to item validation. Apart from clearly separating the concerns involved, this approach also offers the potential to build highly optimized trusted third parties with higher processing capacities that are less viable to become bottlenecks.

3.5. Summary

In this section we have studied the fair exchange problem from a theoretical viewpoint. We have shown how to formally model the fairness requirement and how these formalizations can help in the understanding of which system parameters influence the exchange process.

4. EFFICIENT FAIR EXCHANGE PROTOCOLS

In this section we show how different notions of fairness can be realized by combining appropriate program modules to an exchange protocol as shown in Fig. 2. The advantage of this modular approach is that for different scenarios suitable solutions can be composed. These solutions rely on the properties of the exchanged items, the power of a third party, the effort which is acceptable for the exchange, or on other properties like anonymity of the parties. This section is an enhancement of previous work by Vogt *et al.* [9].

4.1. Modular fair exchange protocols

In an exchange protocol at least the two parties A and B are involved. In the case of failures we also require the cooperation of a trustee T. The party A has an item i_A and B possesses the item i_B.

In order to ease presentation we do not consider compensation as a method to gain fairness. It can however be incorporated into our approach in a straightforward manner. At any point in time during the execution of the basic exchange modules (M_1, M_2, and M_3) an honest participant may become impatient and want to enforce termination of the exchange. In this case the user will interrupt the execution of the protocol and — depending on its current state and the properties of items — directly invoke modules M_4 or M_5 for conflict resolution. Similarly, a protocol run is aborted or resolved if a message is received which indicates that the other party is misbehaving (e.g., a message which contains the wrong information or an invalid signature).

4.1.1. Module Definitions

Module M_1: Negotiate

In a first step A and B negotiate about the exchange. They have to agree on a formal description of each other's item which enables them to verify whether the item received during the exchange protocol is the one which was expected. When both parties know which items shall be exchanged, they also agree on which fair exchange protocol should be used and which modules are used in order to implement it. Additionally, they agree on the name of the trustee possibly involved. After completion of "Negotiate" the exchange itself can be started.

Module M_2: Prepare to exchange

Before the exchange can be started the participating parties have to make sure that conflict resolution will be possible. For generatable items it has to be checked, if they are really generatable[4] as agreed on in module M_1. This means that if the exchange protocol relies

[4]Note that contrary to generatability the revocability of an item needs not be checked in advance, because only the party that provides the revocable item may be disadvantaged.

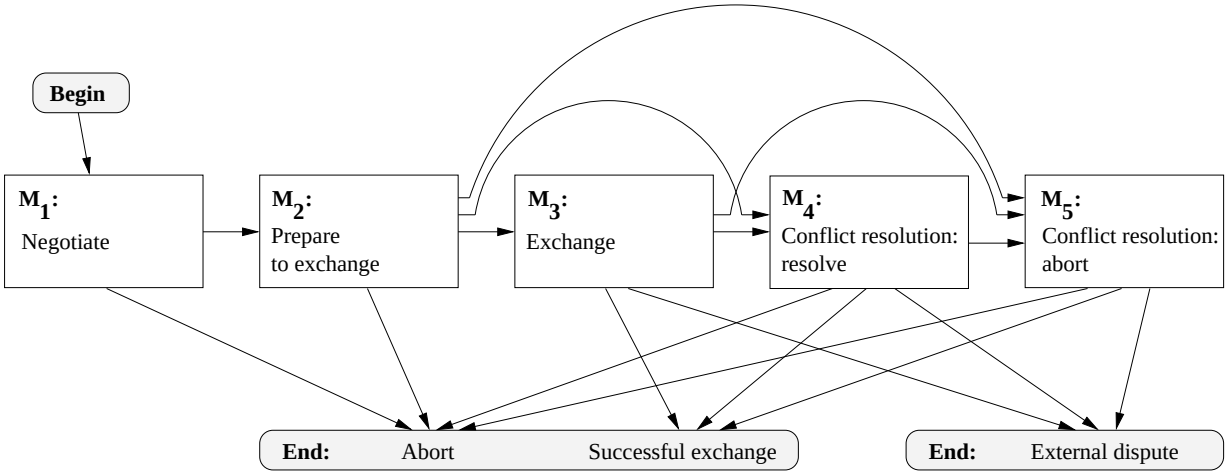


FIGURE 2. Composing fair exchange protocols by using the modular approach

on the generatability of an item, then the party which expects this item has to verify the generatability. If this check fails, it is still safe for the party which has not sent anything yet to stop executing the protocol. A party which has already sent useful information should start M_5 to abort the exchange.

Module M_3: Exchange
The items are now exchanged between the participating parties. In active protocols the trustee can simply deliver the items. If the exchange protocol is optimistic, the following two rules have to be observed:

1. The owner of a generatable item should wait for the other item before he sends his item. This enables him to first check if the other item meets the description.
2. If an item is revocable, its owner should send it first, as he will be able to revoke it, if the other party doesn't send the expected item.

The exchange may either end with both parties possessing the expected items or in the case of a failure with (at least) one party receiving nothing valuable. In the first case fairness F_6 is achieved, so that the protocol can terminate at this point. In the second case fairness must be re-established in one of the subsequent steps. One possible solution is to start an external dispute which however guarantees only a lower degree of fairness (F_1, F_2, or F_3). Alternatively, the modules M_4 and M_5 can be used if the items are generatable or revocable.

Module M_4: Conflict resolution: resolve
This module can only be used by a party that expects the trustee to provide the item in exchange for his own item. If the desired item provides strong generatability, the trustee will succeed in generating the item, thus establishing fairness F_6. If the item is only weakly generatable, the trustee may either succeed or fail. In the first case the protocol terminates with fairness F_6, but in the second case additional steps are necessary: An external dispute can be started. Alternatively, a further attempt to achieve F_6 fairness can be made by executing module M_5. We note that in most cases with weakly generatable items the generation will not fail, as any misbehavior is provable to other parties (e.g., an external arbiter) and thus serves as a proof against the misbehaving party.

Module M_5: Conflict resolution: abort
This module is used by a party which wants to abort the exchange. This can be achieved by either exploiting the revocability of an item or by preventing that the trustee delivers a missing item. In contrast to module M_4 which tries to "force" both parties into a successful exchange, this module aims at "resetting" the exchange so that neither party receives anything valuable.

In case of revocable items the trustee will revoke the item for its provider, if this party can ensure that he has been following the exchange protocol correctly so far and was not able to get the expected item from the other party. If the item provides strong revocability, the trustee will revoke the item and thus establish fairness F_6. The same result is achieved if the trustee succeeds in revoking a weakly revocable item. If the revocation of a weakly generatable item fails, fairness has to be established by other means: An external dispute can be started.

This module can be invoked even if items are not revocable, e.g., in cases where a party has sent a generatable item and wants to withdraw it from the exchange before the trustee can generate it. However, in such cases there is always the chance that resetting the exchange fails because the other party has initiated module M_4 beforehand. Then the trustee can always finish the exchange by delivering the stored item.

4.1.2. Discussion

We can now set up fair exchange protocols by combining these modules. This approach is very flexible, because it provides several protocol variations: If after the execution of a module the fair exchange is completed, then subsequent steps using the other modules are not needed. It is also possible to use only a subset of the five modules by simply omitting one or more modules according to paths displayed in Fig. 2. However, the sequence of the modules must not be changed because this would violate their preconditions. Note that the participating parties can follow distinct paths through the diagram, but they must end up in the same termination state, if the protocol ensures F_6 fairness.

If a protocols makes use of modules M_4 and M_5, it would be possible to execute module M_5 before M_4, as both modules try to achieve the same degree of fairness. However, it is the aim of module M_4 to successfully finish the exchange, while M_5 only tries to roll back the exchange. Thus, we recommend to always execute M_4 first, because it tries to fulfill the effectiveness requirement even in the case of failures.

4.2. Module implementation

In this section, we provide a set of sample module implementations for the scenario where a customer C and a vendor V want to exchange *payment* and *product*.

Each implementation block begins with a description of its preconditions like special item properties given in an "**Assumes:**" clause. We then use the notation $\langle\text{event}\rangle : \langle\text{description}\rangle$ to describe the individual steps of the implementation, where $\langle\text{event}\rangle$ can be sending a message from participant X to Y (designated by $X \to Y$) or some local computation of a participant (designated by its name). The $\langle\text{description}\rangle$ is a brief explanation of the type of message sent or the type of actions performed locally. We denote encryption and decryption functions using key x by e_x and d_x, respectively. We use the capital letters D_X and E_X whenever an asymmetric cryptosystem is applied for party X. A participant can also produce digital signatures. To ease reading, we will abbreviate signing a message m by X and obtaining a signature s by $s := sign_X(m)$. Finally, we also assume the availability of a strongly collision-free cryptographic hash function h. The cryptographic prerequisites are well-studied in the literature [4].

We now provide implementations for the modules specified in Section 4.1.1. In some cases we provide an alternative implementation for a single module. We show later in Section 4.3 how to assemble these implementations to fair exchange protocols.

4.2.1. Implementation of module M_1

A straightforward way to implement the negotiation module is the following:

Module implementation I_1

Assumes: —

$C \to V$	:	desc_{prod}, T, set of possible protocols supported by C
V	:	choose a protocol
$V \to C$	:	desc_{pay}, T, chosen protocol

The customer and the vendor exchange the description of the items which they want to receive and agree on a trustee T which can possibly be invoked. The customer also proposes exchange protocols that are acceptable to him and the vendor dynamically chooses one of these protocols with which he wants to proceed.

For the following implementations it is assumed that for every message it is known to which protocol, which run of a certain protocol, and which step of the protocol it belongs. This prevents attacks that may be constructed by mixing steps from different protocols.

4.2.2. First implementation of module M_2

The "Prepare to exchange" module can be implemented like this:

Module implementation I_{2a}

Assumes: The item descriptions, T, and the protocol have been agreed upon

$C \to V$	:	order product
V	:	choose a random key R
		encrypt product with R, i.e.,
		$EP := e_R(\text{product})$
		compute hash $H := h(EP)$
		encrypt R for the trustee, i.e.,
		$R_T := E_T(R)$
		$S_V := sign_V(\text{desc}_{prod}, \text{desc}_{pay}, T, H, R_T)$
$V \to C$	:	S_V, EP, R_T
C	:	compute hash $H := h(EP)$
		verify signature S_V

In this implementation the customer receives the product encrypted with a random key R, the key R encrypted with the trustee's public key, and a signature from the vendor to commit on this exchange. Note that this signature is assumed to have no meaning outside the fair exchange protocol. This can be guaranteed, if the use of the corresponding signature key is restricted to fair exchange protocols only.

The main idea of this implementation is that the further exchange process is reduced to the exchange of payment and R. Furthermore, the trustee will be able to compute R, if the vendor has sent the correct value R_T. This results in weak generatability for the product as explained in Section 2.3.1.

During execution of this module, it is still possible to abort the exchange if, for example, the signature verification has revealed a bad signature.

4.2.3. Second implementation of module M_2
The implementation above is specifically designed for optimistic fair exchange protocols. For an exchange involving an active trustee, the module M_2 can be implemented in the following manner.

Module implementation I_{2b}

Assumes: T and the protocol have been agreed upon

$C \to T$	:	payment, desc_{prod}
$V \to T$	:	product, desc_{pay}
T	:	check payment, check product versus desc_{prod}

Because the trustee possesses both, payment and product, he can check in advance, if these items match their description. If one of the checks fails, the exchange will be aborted without losing fairness.

During the execution of this module, it is possible that an abort request arrives from one of the participating parties (e.g., because such a party invoked an implementation of module M_5 asynchronously). In this case, T can decide to abort the exchange as it has not yet sent anything to either of the parties. This behavior still satisfies fairness F_6.

4.2.4. Third implementation of module M_2
A variation of implementation I_{2a} is now given where the payment is "made" weakly generatable. Prior to the execution of this module the vendor must have forwarded his product and its description to the trustee, who then checked that the product matches the given item description. As a reply the vendor received the encryption of the product $EP := e_R(product)$, the corresponding random symmetric key R, and the trustee's signature $S_T := sign_T(\text{desc}_{prod}, EP)$. Using this signature the vendor can now prove that his product is strongly generatable by the trustee as we already explained in example 2 in Section 2.3.1.

Module implementation I_{2c}

Assumes: The item descriptions, T, and the protocol have been agreed upon. The trustee ensured that the product is strongly generatable.

C	:	encrypt payment for trustee, i.e., $EM := E_T(payment)$ $S_C := sign_C(\text{desc}_{prod}, \text{desc}_{pay}, T, EM)$
$C \to V$	:	EM, S_C
V	:	verify signature S_C sign the generatable encrypted product, i.e., $S_V := sign_V(\text{desc}_{prod}, \text{desc}_{pay}, T, EP, S_T, EM)$
$V \to C$	:	EP, S_T, S_V
C	:	verify signatures S_T and S_V

Customer C makes his payment weakly generatable by encrypting it with the public key of the trustee. The encrypted payment together with the customer's signature S_C is sent to V, who verifies this signature. If this check fails, the vendor can simply abort the exchange. Otherwise, V returns the information for generating the product, namely the encrypted product, the trustee's signature ensuring generatability of the product, and the vendor's signature which links the product to the payment. C verifies both signatures S_T and S_V and only proceeds if both are correct. If he detects some failure, he needs to abort the exchange using module M_5, as only this prevents V from finishing the exchange with module M_4.

4.2.5. First implementation of module M_3
The exchange module can be implemented like this:

Module implementation I_{3a}

Assumes: C has the encrypted product

$C \to V$	:	payment
V	:	check payment
$V \to C$	:	R
C	:	decode product, i.e., product $:= d_R(EP) = d_R(e_R(\text{product}))$ check product against desc_{prod}

As mentioned earlier, this type of exchange is called optimistic [1], because no third party is required unless a conflict occurs. As most exchange processes can be assumed to run without failures, optimistic protocols can substantially reduce the load that is put on the third party. If a conflict occurs, the disadvantaged party must decide which actions should be used in order to re-establish fairness.

4.2.6. Second implementation of module M_3
If the exchange should be performed with an active trustee, an alternative implementation should be used for module M_3. Because the trustee has already checked both items, he performs the exchange by simply forwarding the items to the waiting parties. If one of the parties does not receive the desired item, he asks the trustee to resend it.

Module implementation I_{3b}

Assumes: T has checked product and payment, exchange not aborted

$T \to C$	:	product
$T \to V$	:	payment

4.2.7. First implementation of module M_4
Module M_4 is a solution for re-establishing fairness by finishing the exchange, if the previous modules failed to achieve fairness. In case the product is generatable it can be implemented like this:

Module implementation I_{4a}

Assumes: C has the encrypted product and the decryption key R_T

$C \to T$	:	desc_{prod}, desc_{pay}, payment, H, R_T, S_V
T	:	verify signature S_V
		decode key: $R := D_T(R_T)$
		check payment, deposit payment
$T \to C$	:	R
C	:	decode product, i.e.,
		product $:= d_R(EP) = d_R(e_R(\text{product}))$
		check product against desc_{prod}

The trustee decodes R, which he sends to the customer in exchange for the payment. The product is decoded by the customer, so that he has to check it by himself. This implementation obviously relies on the vendor, who has to provide the correct values, so that the trustee can generate the correct key R. Since the vendor might have sent incorrect values, actually weak generatability of the product is provided.

4.2.8. Second implementation of module M_4

For an exchange of two generatable items we must distinguish whether C or V invokes the module.

Module implementation I_{4b} for C

Assumes: C has generatable product EP and the signatures of the trustee and vendor (i.e., S_T, S_V), exchange not aborted

$C \to T$	:	$\text{desc}_{prod}, \text{desc}_{pay}, EM, EP, S_T, S_V$
T	:	check S_T and S_V
		decrypt payment, i.e.,
		payment $:= D_T(EM)$
		check payment
		store payment
		recover stored R corresponding to EP
$T \to C$	:	R
C	:	decode product, i.e.,
		product $:= d_R(EP) = d_R(e_R(\text{product}))$

The product is assumed to be strongly generatable so that the trustee always succeeds in generating it for the customer.

For the vendor, the protocol looks slightly different:

Module implementation I_{4b} for V

Assumes: V has weakly generatable money EM and the customer's signature S_C, exchange not aborted

$V \to T$	:	$\text{desc}_{prod}, \text{desc}_{pay}, product, EM, S_C$
T	:	check product
		generate payment by decrypting EM or retrieving a stored payment
		check payment
		store product
$T \to V$	:	payment
V	:	check and deposit payment

As the payment is only weakly generatable, we require T to check it before anything valuable is sent. If the payment turns out to be invalid in I_{4b} for V, the trustee knows that C misbehaved and thus aborts the exchange. Note that for some payment systems it might be necessary to let the trustee immediately deposit the payment at the bank instead of keeping a local copy.

Note that the trustee will not generate anything if the customer has previously succeeded in aborting the protocol with $M5$.

4.2.9. First implementation of module M_5

We can efficiently implement module M_5 as follows: The trustee checks the validity of the weakly generatable product. A detected failure results in revocation of the payment, if it was already sent to the vendor. This guarantees fairness F_6 after execution of this module.

Module implementation I_{5a}

Assumes: The payment is strongly revocable and C has the encrypted product as well as the decryption key R_T

$C \to T$	:	payment, desc_{prod}, desc_{pay}, T, EP, R_T, S_V
T	:	compute hash $H := h(EP)$
		verify signature S_V
		check payment
		decode key $R := D_T(R_T)$
		decode product, i.e.,
		product $:= d_R(EP) = d_R(e_R(\text{product}))$
		check product versus desc_{prod}:
		if "product OK" then
		T : deposit payment
		$T \to C$: R
		C : decode product, i.e.,
		product $:= d_R(EP)$
		elseif "product not OK and payment was already sent to the vendor"
		then
		T : revoke payment

Compared to the implementation I_{4a} the trustee additionally has to check, whether the product is according to the item description desc_{prod}. In most cases this significantly increases the computations done by the trustee.

4.2.10. Second implementation of module M_5

Even if no item is revocable, module M_5 can be used to prevent the trustee from proceeding further in the exchange, e.g. by generating items and delivering them. If the trustee has already completed the exchange (e.g. in module M_3 or M_4), he resends the desired item.

Module implementation I_{5b}

Assumes: —

```
C → T  :  abort request
T      :  if "exchange has been completed" then
              T → C  :  product
          else
              T      :  remember the exchange as
                        "aborted"
              T → C  :  confirmation for the abort
```

The implementation for the vendor is just the same, except that he receives the payment instead of the product.

4.3. Composing protocols

The module implementations described in the previous section can be assembled in different ways according to the rules displayed in Fig. 2. We now describe possible compositions.

P_1: Fair exchange with active trustee.
The combination of modules $\langle I_1, I_{2b}, I_{3b}, I_{5b}\rangle$ results in the basic active exchange scheme for fairness F_6 which has been used in several protocols [25, 42]. Both parties can force the exchange to terminate by invoking I_{5b}.

P_2: Optimistic fair exchange with weak fairness.
The combination $\langle I_1, I_{2a}, I_{3a}, \text{external dispute}\rangle$ results in an optimistic fair exchange protocol which has been presented and discussed in detail by Asokan [5]. This protocol achieved only weak fairness F_2/F_3 and can re-establish fairness only in an external dispute. Special item properties like the weak generatability of the product are not automatically exploited by the protocol. Instead these properties only support the external dispute.

P_3: Optimistic fair exchange with weakly generatable products.
In contrast to protocol P_2, the combination $\langle I_1, I_{2a}, I_{3a}, I_{4a}, \text{external dispute}\rangle$ exploits the weak generatability of the product and can thus achieve fairness F_6 in cases where P_2 relies on an external dispute. However, since this item is only weakly generatable, P_3 may also need an external dispute, if the vendor misbehaves. Thus the protocol achieves only weak fairness F_2/F_3 in general. The advantage of P_3 is that it first makes an attempt to re-establish fairness automatically inside the system. Only if this fails, an external dispute is started.

P_4: Optimistic fair exchange with strongly revocable payments.
The combination $\langle I_1, I_{2a}, I_{3a}, I_{5a}\rangle$ of module implementations exploits the revocability of the payment and weak generatability of the product to achieve strong fairness F_6. In contrast to P_3 the trustee can even guarantee fairness, if the vendor cheats by sending an incorrect item. After the trustee detects such a kind of fraud, he will revoke the payment and thus undo the exchange.

The protocol guarantees only the weaker notion of termination to the vendor. After sending the encrypted product in I_{2a} the vendor has to wait for the customer to proceed. This is not a disadvantage for the vendor, as he can quit executing the protocol and will still receive the payment as soon as the customer will finish the exchange with I_{5a}. A detailed description of this optimistic fair protocol has been published in [6].

P_5: Optimistic exchange with strongly and weakly generatable items
The combination $\langle I_1, I_{2c}, I_{3a}, I_{4b}, I_{5b}$ for $C\rangle$ is an instantiation of the optimistic fair exchange protocols proposed by Asokan, Shoup and Waidner [7]. Due to the strong generatability of the product and the weak generatability of the payment an external dispute can always be avoided.

In I_{2c} the vendor may simply quit, if he detects a failure or if he is not willing to wait longer. If the customer wants to resolve a conflict in I_{2c}, he has to start I_{5b} which instructs the trustee not to proceed with the protocol. During the execution of I_{3a} the customer relies on I_{4b} to resolve conflicts by exploiting the strong generatability of the product. This guarantees fairness F_6 and termination for the customer.

If the vendor does not receive the payment in I_{3a}, he consequently does not send R to C and starts I_{4b} instead. Then the trustee tries to finish the exchange by generating the payment and storing the product for the customer. An interesting case arises if the customer not yet executed I_{4b} and the trustee fails to generate the payment because C encrypted wrong values. As the product has not yet been sent to C, the trustee answers the customer's attempt to cheat by simply aborting this exchange. This ensures fairness F_6 and termination for V.

P_6: Efficient optimistic fair exchange with strongly revocable payments.
The combination $\langle I_1, I_{2a}, I_{3a}, I_{4a}, I_{5a}\rangle$ is a very efficient optimistic fair exchange protocol which was previously presented by Vogt *et al.* [9]. Due to the strong revocability of the payment it achieves fairness F_6 similar to the protocol P_4. However, this protocol tries to reduce the participation of the trustee as far as possible. If the customer did not receive the decryption key for the product during module M_3, the trustee first in I_{4a} tries to generate the decryption key without checking whether the decryption of the product works correctly or not. It is assumed that in most cases this simple and efficient implementation I_{4a} is sufficient to establish fairness. Thus the overhead of executing an implementation of module M_5 is avoided.

Only if the decryption of the product fails, the customer will execute I_{5a}. There the trustee additionally checks the correctness of the product, which might be quite costly, and revokes the payment in the case of a failure. This finally guarantees fairness F_6 under all circumstances.

However, similar to P_4 this protocol guarantees only the weaker form of termination to V, since V may only be informed about a successful completion of the exchange if C starts a conflict resolution which results in T depositing the payment.

P_7: Fair exchange with active participation of a lightweight trustee.
Combination $\langle I_1, I_{2a}, I_{4a}, \text{external dispute}\rangle$ is another implementation which uses the active participation of a trustee in every exchange. However the capabilities of the trustee are limited to the exchange of the payment against the decryption key of the product. But as the decrypted product might still turn out to be incorrect, the protocol can only achieve weak fairness F_2/F_3 in an external dispute.

The NetBill payment protocol [43] uses a similar idea to ensure fairness.

P_8: Another fair exchange variant with active trustee.
Module combination $\langle I_1, I_{2a}, I_{5a}\rangle$ is a variation of protocol P_1. After the exchange has been prepared in I_{2a}, an active trustee is used to finally perform the swap of product and payment in I_{5a}. It should be noted that although the implementation I_{5a} is executed it is not necessary to use a payment system with revocability. The reason for this is that the payment was not sent to the vendor during the previous steps and hence it does not need to be revoked if a failure is detected by the trustee.

4.4. Discussion

Table 1 gives an overview of the composed protocols. In the table we have also noted the item properties assumed by the protocols and the fairness guarantees achieved.

The protocol modules (not their implementation) can be regarded as an abstraction of the general interaction pattern of fair exchange protocols. This is why it is possible (in contrast to other work [44, 45]) to not only model optimistic protocols in our framework, but also those involving an active TTP as well as optimistic protocols with revocable items. Well-known protocols (such as other optimistic ones by Boyd and Foo [18], Garay, Jakobsson and MacKenzie [46] or Markowitch and Saeednia [44]) all follow the same interaction pattern and so can be formulated within our framework (with different module implementations). As it has been demonstrated in this paper, formulating existing protocols (like the ones mentioned above) within our protocol framework may reveal new possibilities of module combinations leading to improved exchange protocols. Note that the framework was designed to incorporate asynchronous protocols as well as synchronous ones. The only difference here is that the notion of a party "wanting to abandon the exchange" is defined in terms of real-time bounds.

Not all fair exchange protocols can be adapted to our framework. For example, we do not see how *gradual exchange protocols* (which rely on special assumptions and achieve a different notion of fairness) can be formulated as a composition of our protocol modules. We will discuss these protocols in the context of related work in the following section.

5. RELATED WORK

The problem of fair exchange has been studied under many different headings and from many different perspectives. For example, contract signing, key exchange and certified mail all share aspects of the problem.

Sometimes, fair exchange has been mentioned in the context of protocols for *non-repudiation* [22, 47–49]. The property of non-repudiation ensures that participants collect evidence so that the other party cannot deny that a particular event has taken place. For example, *non-repudiation of receipt* means that a party has obtained a proof that the other party has received a particular message. In some sense, non-repudiation can be achieved using a fair exchange protocol with one item being a receipt for the other item. However, in contrast to the general definition of fair exchange, there is a dependency between the exchanged items, because, like in certified email, the receipt must contain an explicit reference to the original message (e.g., its hash value). Most existing fair exchange protocols can be augmented to provide non-repudiation. However, it should be clear that a *minimal* fair exchange protocol (i.e., a protocol which satisfies the effectiveness, timeliness and fairness properties of Section 2 and *no other properties*) is not *per se* powerful enough to provide this feature [5]. Another question is whether it is possible to build fair exchange protocols by using protocols for non-repudiation (like those for certified mail [50–52]). Fair exchange can be implemented by letting both parties separately exchange their items, each of them using a non-repudiation protocol. Using such an approach, strong fairness cannot be achieved because it cannot be guaranteed that both protocol parts are executed atomically. However, the non-repudiation proofs can be used in an external dispute to establish weak fairness.

Atomicity was identified as a generic property of e-commerce protocols by Tygar [43]. The understanding is closely related to the notion of an atomic transaction from the field of databases and comprises all the usual properties of fair exchange including non-repudiation. Viewing fair exchange as a distributed transaction has conceptual advantages since it offers an analogy to a

TTP	properties of payment	properties of product	fairness for customer	fairness for vendor	protocol/ references
active	*	*	F_6	F_6	P_1 [25, 42], P_8
active	*	WG	F_2/F_3	F_2/F_3	P_7 [43]
optimistic	*	WG	F_2/F_3	F_2/F_3	P_2 , P_3 [5]
optimistic	SR	WG	F_6	F_6, α	P_4 [6], P_6 [9]
optimistic	WG	SG	F_6	F_6	P_5 [7]

* = none
SG = strongly generatable
WG = weakly generatable
SR = strongly revocable
α = with "weak" termination

TABLE 2. Summary of the protocol compositions and achievable fairness levels depending on item properties.

concept which is quite well understood [53]. Also, atomic commitment protocols like "two-phase commit" (2PC) [54] can be adapted quite easily to enable fair exchange. Consequently, the transactional view has lead to fair exchange protocols requiring transaction coordinators which in terms of fair exchange resemble an active trusted third party [43, 55, 56]. This rather implementation centric view can be regarded as a disadvantage of the transactional approach to fair exchange. As discussed in Section 2, the transactional view also slightly obscures the security aspect of the transaction [56]. An advantage of the transactional view of fair exchange probably is its straightforward methodological support for *multi-party fair exchange* [39, 55, 56]. In multi-party fair exchange more than two parties want to fairly exchange items with each other. This resembles the situation in which a customer has filled a virtual shopping basket with closely related items from different vendors and wants to pay electronically for all items. The standard example consists of a customer who wants to book a flight, rent a hotel room and a hire a car for a business trip: Any missing item renders the other items useless.

The transactional view can be regarded as a continuation of the early basic research on fair exchange because the earliest fair exchange protocols that have appeared in literature involve the active participation of a trusted third party in every run. Such protocols have been presented by Bürk and Pfitzmann [42] and by Franklin and Reiter [25]. However, requiring the active participation of a trusted third party in every exchange has some obvious drawbacks (such as the potential performance bottleneck or the need for permanent availability). These drawbacks can be partly circumvented by *optimistic* fair exchange protocols [1, 7, 30]. In optimistic exchange protocols both participating parties try to handle the exchange on their own and only call for the participation of a trusted third party if something went wrong during the exchange. If the protocol is known to ensure fairness both parties are aware that they cannot gain an advantage by acting maliciously. Therefore, the situation in which the assistance of the trustee is required is not likely to happen in practice. Recently, the optimistic approach has also been extended to multi-party fair exchange [57–59].

The fairness level of optimistic exchange protocols depends on whether items are generatable, revocable or the like. If items do not possess such special properties, only weak fairness is achievable. Postulating that an item is revocable is usually only justified for a small class of items, e.g., payments or certain types of privileges. This could be an explanation for the fact that most of the work in this area concentrates on generatable items and how to make items generatable, e.g., signatures [12, 16, 18, 19].

Some protocols try to avoid the use of a trusted third party, for example, by relying on special notions like a publicly visible *blackboard* [60] or the use of the existing Internet infrastructure [52]. In any case, to resolve conflicts, a trusted authority is required either inside or outside of the system. Which of these notions can correctly be called a trusted third party depends on the understanding of this term. Other authors weaken the definition of fairness further and only provide strong incentives to behave correctly [61–63].

The items in consideration for fair exchange can also be "continuous" ones such as the provision of a communication link. For the latter type of service *gradual exchange* protocols [64, 65] can be used. The basic idea behind gradual exchange is to repeatedly grant small low-value portions of the services. Hence, interrupting the exchange can only lead to one party gaining a small advantage over the other. This minimizes the amount of "unfairness" which a participating party may experience. A precondition for gradual exchange protocols is that the services must be divisible into parts with "near-to-equal" value. Obviously, the smaller these parts become the more the communication overhead increases. Conversely, when splitting the service into larger parts there is a non-negligible risk of loss in case the protocol is interrupted. In order to avoid this situation a different protocol for fairly exchanging the individual parts

is required. For services or items which are not divisible such a protocol must be used anyway.

In the context of key exchange and contract signing other concepts for fair exchange without a trustee have been proposed: The first one is called gradual release of a secrets and has led to several protocol proposals [61, 66–70]. Usually two keys, which are mostly used for decrypting signatures, are exchanged bit by bit in several rounds. Then both parties always have approximately the same knowledge about the decryption key. If the communication is disrupted, the remaining unknown bits can be computed by a brute force search, which in principle enables both parties to complete the exchange alone. However, it is essential that both parties have near to equal computing power so that they have to invest about the same amount of time and money to complete the exchange. Another limitation of this kind of exchange is that nobody knows, if the other party is willing to complete an interrupted exchange by executing a brute force key search. Thus the exchange can be unfair, if one party does not complete the exchange while the other one does.

A different solution for an exchange without a trustee builds on a probabilistic fairness definition [71–73]: The probability that one party receives something while the other one receives nothing is reduced to $1/n$, where n is a security parameter which is approximately the number of communication rounds. Basically, this means that the probability of unfairness can be made arbitrarily small. However protocols achieving this kind of fairness rely on special item properties (e.g., contracts of a certain form [71, 72]) or unusually system assumptions (e.g., that the communication channels have a limited time delay [73]) and thus cannot be regarded as being practical.

In 1999, Garay, Jakobsson and MacKenzie [46] introduced the notion of *abuse-freeness* in the context of contract signing. A protocol is abuse-free if at any time during protocol execution no party has the power to prove to a third party that it can determine the outcome of the protocol. For example, Alice may negotiate an electronic employment contract with Bob which Bob has signed already and sent to Alice. But Alice does not actually want to work for Bob. Her aim is to negotiate a better contract with Claire. Since Alice has the power to determine the outcome of the exchange with Bob, she can show this contract to Claire and demand a higher salary. Protocols which are abuse-free avoid this problem. Abuse-freeness has also been studied in the context of multi-party contract signing [59, 74] and has been the subject of formal analysis [32, 75–77].

The general area of contract signing has been the focus of other work in formal analysis of exchange protocols [32, 36]. Formal analysis is particularly difficult because fair exchange not only involves an aspect of atomicity but also an aspect of security. Formal analysis also heavily depends on the system model and there are many system parameters (synchrony, communication primitives, etc.) which have not been studied in this context yet. Tool supported analysis is only possible in scenarios which often oversimplify the aspect of security [36, 78]. As argued earlier, standard cryptographic definitions of security lead to non-discrete system models where proofs must be done by hand [30].

A recent line of research has investigated hardware support for fair exchange [23, 24, 65, 79], where trust is partly substituted by tamper-proof hardware devices that are carefully designed to ensure fairness.

6. CONCLUSIONS

Fair exchange is a problem of substantial practical significance in electronic commerce. Products, payments, and services must be exchanged fairly to ensure the continuing growth of the electronic marketplace. In order to increase the trust that participating parties place in exchange services it is important to state precisely the guarantees of different protocols with respect to fairness and efficiency.

In this article we have presented a comprehensive study of fair exchange. Firstly, we have studied the problem from a practical viewpoint, focusing on intuitive definitions of fairness and a discussion of other factors that influence the exchange process. Secondly, we have presented a survey of formal definitions of fair exchange.

Thirdly, we have presented a useful factorization of the fair exchange problem, i.e., we have analyzed the exchange process and separated the functionality of different phases into separate protocol modules. Combining some of the possible implementations of these modules results in a multitude of fair exchange protocols. This helps to understand and compare these solutions with respect to their efficiency and the degree of fairness they offer. Finally, we have extensively surveyed the literature on fair exchange.

A better understanding of fair exchange has many practical consequences. For example, customers and vendors can now select a certain protocol that suits the application or situation needs best. For example, if products of considerable value (like a new CAD-program) are exchanged, both parties will probably favor a protocol which guarantees a strong fairness $F_4/F_5/F_6$ even if this comes at a higher cost (because they might have to pay a trustee). On the other hand, both parties might be willing to agree on a weakly fair $F_1/F_2/F_3$ protocol if they simply exchange the latest football results.

Moreover, by our compositional approach it is now possible to select exchange protocols for a given level of fairness and given item properties dynamically. In practical settings this enables a customer to choose an item from an on-line catalog, select the desired level of fairness and have the rest of the exchange be executed automatically; it is no longer necessary for the user to select a specific protocol which clearly is an important

step towards more user friendliness.

ACKNOWLEDGEMENTS

We wish to thank Levente Buttyán for his comments on an earlier version of this paper. We also thank Heiko Mantel for helpful discussions and the anonymous referees for their detailed comments. This work was performed while all authors were affiliated with the Department of Computer Science of Darmstadt University of Technology.

REFERENCES

[1] Asokan, N., Schunter, M., and Waidner, M. (1997) Optimistic protocols for fair exchange. Matsumoto, T. (ed.), *4th ACM Conference on Computer and Communications Security*, Zürich, Switzerland, Apr., pp. 6–17, ACM Press, New York.

[2] Basu, A., Charron-Bost, B., and Toueg, S. (1996) Simulating reliable links with unreliable links in the presence of process crashes. *Proceedings of the 10th International Workshop on Distributed Algorithms (WDAG96)*, Bologna, Italy, Oct., pp. 105–122, Springer-Verlag.

[3] Schneider, F. B. (1993) What good are models and what models are good? Mullender, S. (ed.), *Distributed Systems*, chap. 2, pp. 17–26, Addison-Wesley, Reading, MA, second edn.

[4] Menezes, A. J., Oorschot, P. C. V., and Vanstone, S. A. (1997) *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL.

[5] Asokan, N. (1998) *Fairness in electronic commerce*. Ph.D. thesis, University of Waterloo, Canada.

[6] Pagnia, H. and Vogt, H. (1999) Exchanging goods and payment in electronic business transactions. *Proceedings of the Third European Research Seminar on Advances in Distributed Systems (ERSADS)*, Madeira Island, Portugal, Apr.

[7] Asokan, N., Shoup, V., and Waidner, M. (1998) Asynchronous protocols for optimistic fair exchange. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May, pp. 86–99, IEEE Computer Society Press, Los Alamitos, CA.

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

[9] Vogt, H., Pagnia, H., and Gärtner, F. C. (1999) Modular fair exchange protocols for electronic commerce. *Proceedings of the 15th Annual Computer Security Applications Conference*, Phoenix, Arizona, Dec., pp. 3–11, IEEE Computer Society Press, Los Alamitos, CA.

[10] Gärtner, F. C., Pagnia, H., and Vogt, H. (1999) Approaching a formal definition of fairness in electronic commerce. *Proceedings of the International Workshop on Electronic Commerce (WELCOM '99)*, Lausanne, Switzerland, Oct., pp. 354–359, IEEE Computer Society Press, Los Alamitos, CA.

[11] Wilhelm, U. G. (1999) *A Technical Approach to Privacy based on Mobile Agents protected by Tamper-resistant Hardware*. Ph.D. thesis, École Polytechnique Fédérale de Lausanne, Switzerland.

[12] Mao, W. (1997) Verifiable escrowed signature. *Information Security and Privacy – ACISP '97*, Sydney, Australia, Jul., vol. 1270 of *Lecture Notes in Computer Science*, pp. 240–248, Springer-Verlag, Berlin.

[13] Asokan, N., Shoup, V., and Waidner, M. (2000) Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, **18**, 593–610.

[14] Asokan, N., Shoup, V., and Waidner, M. (1998) Optimistic fair exchange of digital signatures. Nyberg, K. (ed.), *Advances in Cryptology – EUROCRYPT '98*, Espoo, Finland, Jun., pp. 591–606, Lecture Notes in Computer Science, Springer-Verlag, Berlin.

[15] Camenisch, J. and Damgård, I. (1998) Verifiable encryption and applications to group signatures and signature sharing. Tech. Rep. RS-98-32, BRICS, Department of Computer Science, Aarhus University, Denmark.

[16] Ateniese, G. (1999) Efficient verifiable encryption (and fair exchange) of digital signatures. *Proceedings of 6th ACM Conference on Computer and Communications Security (CCS '99)*, Singapore, Nov., pp. 138–146, ACM Press, New York.

[17] Bao, F., Deng, R. H., and Mao, W. (1998) Efficient and practical fair exchange protocols with off-line TTP. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May, pp. 77–85, IEEE Computer Society Press, Los Alamitos, CA.

[18] Boyd, C. and Foo, E. (1998) Off-line fair payment protocol using convertible signatures. *Advances in Cryptology – ASIACRYPT '98*, Beijing, China, Oct., vol. 1514 of *Lecture Notes in Computer Science*, pp. 271–285, Springer-Verlag, Berlin.

[19] Chen, L. (1998) Efficient fair exchange with verifiable confirmation of signatures. Ohta, K. and Pei, D. (eds.), *Advances in Cryptology – ASIACRYPT '98*, Beijing, China, 18–22 Oct., vol. 1514 of *Lecture Notes in Computer Science*, pp. 286–299, Springer-Verlag, Berlin.

[20] Boyar, J., Chaum, D., Damgård, I. B., and Pedersen, T. P. (1990) Convertible undeniable signatures. Menezes, A. J. and Vanstone, S. A. (eds.), *Advances in Cryptology – CRYPTO '90*, Santa Barbara, CA, Aug., pp. 189–205, no. 537 in Lecture Notes in Computer Science, Springer-Verlag, Berlin.

[21] Chaum, D. (1983) Blind signatures for untraceable payments. *Advances in Cryptology – CRYPTO '82*, Santa Barbara, CA, Aug., pp. 199–203, Plenum, New York.

[22] Zhou, J. (1996) *Non-repudiation*. Ph.D. thesis, University of London.

[23] Vogt, H., Pagnia, H., and Gärtner, F. C. (2001) Using smart cards for fair exchange. *Electronic Commerce – WELCOM 2001*, Heidelberg, 16–17 Nov., vol. 2232 of *Lecture Notes in Computer Science*, pp. 101–113, Springer-Verlag, Berlin.

[24] Vogt, H., Gärtner, F. C., and Pagnia, H. (2002) Supporting fair exchange in mobile environments. *ACM/Kluwer Journal on Mobile Networks and Applications (MONET)*, to appear.

[25] Franklin, M. K. and Reiter, M. K. (1997) Fair exchange with a semi-trusted third party. Matsumoto, T. (ed.), *4th ACM Conference on Computer and Communications Security*, Zürich, Switzerland, Apr., pp. 1–5, ACM Press, New York.

[26] Denning, D. E. (1976) A lattice model of secure information flow. *Communications of the ACM*, **19**, 236–243.

[27] Buttyàn, L. (2001) *Building blocks for secure services: Autenticated key transport and rational exchange protocols*. Ph.D. thesis, École Polytechnique Fédérale de Lausanne, no. 2511.

[28] Morris, P. (2000) *Introduction to Game Theory*. Springer-Verlag, Berlin.

[29] Pfitzmann, B., Schunter, M., and Waidner, M. (2000) Secure reactive systems. Research Report RZ 3206 (#93252), IBM Research.

[30] Schunter, M. (2000) *Optimistic Fair Exchange*. Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany.

[31] Cervesato, I., Durgin, N., Lincoln, P., Mitchell, J., and Scedrov, A. (1999) A meta-notation for protocol analysis. *Proceedings of the 12th Annual IEEE Computer Security Foundations Workshop – CSFW'99*, Mordano, Italy, Jun., pp. 55–69, IEEE Computer Society Press, Los Alamitos, CA.

[32] Chadha, R., Kanovich, M., and Scedrov, A. (2001) Inductive methods and contract-signing protocols. Samarati, P. (ed.), *Proceedings of the 8th ACM Conference on Computer and Communication Security*, Philadelphia, PA, Nov., pp. 176–185, ACM Press, New York.

[33] Lamport, L. (1977) Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, **3**, 125–143.

[34] Alpern, B. and Schneider, F. B. (1985) Defining liveness. *Information Processing Letters*, **21**, 181–185.

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

[36] Shmatikov, V. and Mitchell, J. C. (2002) Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, **283**, 419–450.

[37] Schneider, F. B. (2000) Enforceable security policies. *ACM Transactions on Information and System Security*, **3**, 30–50.

[38] Even, S. and Yacobi, Y. (1980) Relations amoung public key signature systems. Tech. Rep. 175, Computer Science Department, Technicon, Haifa, Israel.

[39] Franklin, M. K. and Tsudik, G. (1998) Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. *Financial Cryptography – FC '98*, Anguilla, British West Indies, Feb., vol. 1465 of *Lecture Notes in Computer Science*, pp. 90–102, Springer-Verlag, Berlin.

[40] Aguilera, M. K., Chen, W., and Toueg, S. (2000) Failure detection and consensus in the crash recovery model. *Distributed Computing*, **13**, 99–125.

[41] Sander, T. and Tschudin, C. (1998) Towards mobile cryptography. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May, IEEE Computer Society Press.

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

[43] Tygar, J. D. (1996) Atomicity in electronic commerce. *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC '96)*, Philadelphia, PA, May, pp. 8–26, ACM Press, New York.

[44] Markovitch, O. and Saeednia, S. (2002) Optimistic fair exchange with transparent signature recovery. *Financial Cryptography – FC 2001*, Grand Cayman, British West Indies, 19–22 Feb., vol. 2339 of *Lecture Notes in Computer Science*, pp. 339–350, Springer-Verlag, Berlin.

[45] Liu, P., Ning, P., and Jajodia, S. (2000) Avoiding loss of fairness owing to process crashes in fair data exchange protocols. *Proceedings of the IEEE International Conference on Dependable Systems and Networks, Workshop on Dependability despite Malicious Faults*, New York, Jun., pp. 631–640, IEEE Computer Society Press, Los Alamitos, CA.

[46] Garay, J. A., Jakobsson, M., and MacKenzie, P. (1999) Abuse-free optimistic contract signing. Wiener, M. (ed.), *Advances in Cryptology – CRYPTO '99*, Santa Barbara, CA, 15–19 Aug., vol. 1666 of *Lecture Notes in Computer Science*, pp. 449–466, Springer-Verlag, Berlin.

[47] Zhou, J. and Gollmann, D. (1996) A fair non-repudiation protocol. *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May, pp. 55–61, IEEE Computer Society Press, Los Alamitos, CA.

[48] Zhou, J. and Gollmann, D. (1997) An efficient non-repudiation protocol. *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, Rockport, MA, Jun., pp. 126–132, IEEE Computer Society Press, Los Alamitos, CA.

[49] Zhou, J., Deng, R., and Bao, F. (1999) Evolution of fair non-repudiation with TTP. *Information Security and Privacy – ACISP '99*, Wollongong, Australia, 7–9 Apr., vol. 1587 of *Lecture Notes in Computer Science*, pp. 258–269, Springer-Verlag, Berlin.

[50] Bahreman, A. and Tygar, J. (1994) Certified electronic mail. *Proceedings of the ISOC Symposium on Network and Distributed Systems Security*, San Diego, CA, Feb., pp. 3–19, IEEE Computer Society Press, Los Alamitos, CA.

[51] Zhou, J. and Gollmann, D. (1996) Certified electronic mail. *Computer Security – ESORICS '96*, Rome, Italy, Sep., vol. 1146 of *Lecture Notes in Computer Science*, pp. 160–171, Springer-Verlag, Berlin.

[52] Schneier, B. and Riordan, J. (1998) A certified e-mail protocol. *Proceedings of the 14th Annual Computer Security Applications Conference*, Scottsdale, AZ, Dec., pp. 347–352, IEEE Computer Society Press, Los Alamitos, CA.

[53] Lynch, N. A., Merritt, M., Weihl, W., and Fekete, A. (1994) *Atomic Transactions*. Morgan Kaufmann, San Mateo, CA.

[54] Bernstein, P., Hadzilacos, V., and Goodman, N. (1987) *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA.

[55] Schuldt, H., Popovici, A., and Schek, H.-J. (1999) Give me all I pay for – Execution guarantees in electronic commerce payment processes. *GI Workshop Informatik '99: Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications*, Paderborn, Germany, 6 Oct., Springer-Verlag, Berlin.

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

[57] Bao, F., Deng, R., Nguyen, K. Q., and Varadharajan, V. (1999) Multi-party fair exchange with an off-line trusted neutral party. *Proceedings of the 10th International Workshop on Database & Expert Systems Applications*, Florence, Italy, 1–3 Sep., pp. 858–862, IEEE Computer Society Press, Los Alamitos, CA.

[58] Markowitch, O. and Kremer, S. (2000) A multi-party optimistic non-repudiation protocol. *Information Security and Cryptology – ICISC 2000*, Seoul, Korea, Dec., vol. 2015 of *Lecture Notes in Computer Science*, pp. 109–122, Springer-Verlag, Berlin.

[59] Baum-Waidner, B. and Waidner, M. (2000) Round-optimal and abuse-free multi-party contract signing. *27th International Colloquium on Automata, Languages and Programming (ICALP '2000)*, Geneva, Switzerland, Jul., pp. 524–535, Lecture Notes in Computer Science, Springer-Verlag, Berlin.

[60] Pagnia, H. and Jansen, R. (1997) Towards multiple-payment schemes for digital money. Hirschfeld, R. (ed.), *Financial Cryptography: First International Conference, FC '97*, Anguilla, British West Indies, 24–28 Feb., vol. 1318 of *Lecture Notes in Computer Science*, pp. 203–215, Springer-Verlag.

[61] Syverson, P. (1998) Weakly secret bit commitment: Applications to lotteries and fair exchange. *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW '98)*, Rockport, Massachusetts, Jun., pp. 2–13, IEEE Computer Society Press, Los Alamitos, CA.

[62] Jakobsson, M. (1995) Ripping coins for fair exchange. Guillou, L. C. and Quisquater, J.-J. (eds.), *Advances in Cryptology – EUROCRYPT '95*, St. Malo, France, 21–25 May, vol. 921 of *Lecture Notes in Computer Science*, pp. 220–230, Springer-Verlag, Berlin.

[63] Buttyàn, L. and Hubaux, J.-P. (2001) Rational exchange – A formal model based on game theory. *Electronic Commerce – WELCOM 2001*, Heidelberg, Nov., vol. 2232 of *Lecture Notes in Computer Science*, pp. 114–126, Springer-Verlag, Berlin.

[64] Sandholm, T. W. and Lesser, V. R. (1995) Equilibrium analysis of the possibilities of unenforced exchange in multiagent systems. Mellish, C. S. (ed.), *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, Aug. 20–25, pp. 694–703, Morgan Kaufmann, San Mateo, CA.

[65] Zhou, J. and Lam, K.-Y. (1999) A secure pay-per-view scheme for web-based video service. *Public Key Cryptography – PKC '99*, Kamakura, Japan, Mar., vol. 1560 of *Lecture Notes in Computer Science*, pp. 315–326, Springer-Verlag, Berlin.

[66] Even, S. (1982) A protocol for signing contracts. Gersho, A. (ed.), *Advances in Cryptology: A Report on CRYPTO 81*, Santa Barbara, USA, Aug., pp. 148–153, ECE Report No 82-04, U.C. Santa Barbara, Department of Elec. and Computer Eng.

[67] Blum, M. (1983) How to exchange (secret) keys. *ACM Transactions on Computer Systems*, **1**, 175–193.

[68] Brickell, E. F., Chaum, D., Damgård, I. B., and van de Graaf, J. (1987) Gradual and verifiable release of a secret. *Advances in Cryptology – CRYPTO '87*, Santa Barbara, CA, Aug., vol. 293 of *Lecture Notes in Computer Science*, pp. 156–166, Springer-Verlag, Berlin.

[69] Damgård, I. B. (1993) Practical and provably secure release of a secret and exchange of signatures. Helleseth, T. (ed.), *Advances in Cryptology – EUROCRYPT '93*, Lofthus, Norway, May, vol. 765 of *Lecture Notes in Computer Science*, pp. 200–217, Springer-Verlag, Berlin.

[70] Boneh, D. and Naor, M. (2000) Timed commitments. *Advances in Cryptology – CRYPTO '2000*, Santa Barbara, CA, vol. 1880 of *Lecture Notes in Computer Science*, pp. 236–254, Springer-Verlag, Berlin.

[71] Rabin, M. O. (1983) Transaction protection by beacons. *Journal of Computer and System Science*, **27**, 256–267.

[72] Ben-Or, M., Goldreich, O., Micali, S., and Rivest, R. L. (1990) A fair protocol for signing contracts. *ACM Transactions on Information Theory*, **36**, 40–46.

[73] Markowitch, O. and Roggeman, Y. (1999), Probabilistic non-repudiation without trusted third party. Presented at the Second Conference on Security in Communication Networks (SCN99), Amalfi, Italy, no conference proceedings.

[74] Garay, J. A. and MacKenzie, P. (1999) Abuse-free multi-party contract signing. *Distributed Computing – DISC '99*, Bratislava, Slovak Rep., 27–29 Sep., vol. 1693 of *Lecture Notes in Computer Science*, pp. 151–165, Springer-Verlag, Berlin.

[75] Shmatikov, V. and Mitchell, J. C. (2001) Analysis of abuse-free contract signing. *Financial Cryptography – FC 2000*, Anguilla, British West Indies, 21–24 Feb., vol. 1962 of *Lecture Notes in Computer Science*, pp. 174–191, Springer-Verlag, Berlin.

[76] Das, S. and Dill, D. L. (2001) Successive approximation of abstract transition relations. *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, Boston, MA, Jun., pp. 51–58, IEEE Computer Society Press, Los Alamitos, CA.

[77] Kremer, S. and Raskin, J.-F. (2002) Game analysis of abuse-free contract signing. *Proc. of the 15th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, Canada, Jun., IEEE Computer Society Press, Los Alamitos, CA.

[78] Kremer, S. and Raskin, J.-F. (2000) Formal verification of non-repudiation protocols – A game approach. Tech. Rep. 431, Université Libre de Bruxelles, Belgium, presented at the Workshop on Formal Methods and Computer Security, Chicago, 2000.

[79] Pagnia, H., Vogt, H., Gärtner, F. C., and Wilhelm, U. G. (2000) Solving fair exchange with mobile agents. *ASA/MA 2000*, Zürich, Switzerland, Sep., vol. 1882 of *Lecture Notes in Computer Science*, pp. 57–72, Springer-Verlag, Berlin.