SECURE E-COMMERCE TRANSACTIONS, MODELING AND IMPLEMENTATION ASPECTS

Jian Tang
Department of Computer Science
Memorial University of Newfoundland
St. John's, NF, Canada

Ada Waichee Fu
Department of Computer Science and Engineering
Chinese University of Hong Kong
Shatin, Hong Kong

Abstract The requirement for secure e-commerce transactions arises due to the potentially hostile execution environment where not only hardware can fail, but also the messages are subject to attacks by intruders, and some parties may suffer from negligent or even malicious behaviors of some other parties. Most of the existing work paid attention only to message security. In this paper, we propose a secure e-commerce transaction model. We focus our attention on modeling the security aspects at the transaction level, and discuss the properties an e-commerce transaction protocol must have in order to support the semantics of the transactions. The semantics we use is based in the first instance on the notion of atomicity proposed in [13], and then extended to a broader context where the behaviors of untrustworthy players are taken into consideration

Keywords: e-commerce, transaction, atomicity, dispute, protocol

1. INTRODUCTION

E-commerce transactions nowadays have become a popular way of conducting commercial activities for many business organizations. Compared with the traditional way of doing business, e-commerce transactions have the advantage of high efficiency and simple interface. During the execution of the transactions, the parties involved exchange electronic messages via a communication network, such as the Internet, and make decisions.

The steps the parties follow in the execution are governed by a collection of rules, called protocols. Ideally, the protocols are designed in such a way that during the executions of transactions, people can mutually benefit and their interests can be protected. In other words, an e-commerce transaction should be *secure*. The requirement for secure e-commerce transactions arises due to the potentially hostile environment where the transactions are executed. In this environment, not only

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35658-7_21](https://doi.org/10.1007/978-0-387-35658-7_21)

R. Meersman et al. (eds.), *Semantic Issues in E-Commerce Systems*
© IFIP International Federation for Information Processing 2003

hardware can fail, but also the messages are subject to attacks by intruders, and some parties may suffer from negligent or even malicious behaviors of some other parties.

For example, when a customer is trying to purchase a product through the Internet, how does she know if the person she is contacting is really the one who put the advertisement there at the first place? Is it possible that after she makes the payment she never gets anything back? What provisions should be made in an e-commerce transaction protocol if some parties in the transactions are not trusted? These kinds of anomalies are almost non-existent if customer and merchant do the transactions face to face. Even if the transactions are carried out through other means such as postal mail, phone, or fax, the above problems are not considered to be a threat often. The reasons are twofold. First, customers usually do business indirectly (i.e., through a medium) only with venders they trust, who may be someone they knew about before or introduced by some reliable sources. Second, many transactions involve agreements with hand written signatures. This provides some guarantee that in case something goes wrong, there are legal systems to turn to. With the scale of the activities like that in the Internet, it is hardly possible to know the trustworthiness of a party in advance every time one wants to initiate a transaction. Neither is it realistic to expect human legal systems to solve all the problems that may possibly arise, aside from the fact that handling electronic messages is not nearly as easy and convenient as handling natural languages. Due to these factors, we can expect that secure e-commerce transaction is a more difficult issue than its traditional counter parts.

Although the security of e-commerce transactions has been widely recognized as an important issue and has been the subject of extensive studies in the past several years, most of the existing work paid attention only to message security. The work reported in [1] discusses some security issues at the transaction level such as formulation of dispute claims, an architecture of dispute handling systems, etc. It does not however address the issue of how an overall model is constructed. Neither does it mention how a protocol is able to support dispute handling. The work in [11] proposes an e-transaction model which addresses security issues at both message and transaction level. However, it assumes that the players are trusted so that the protocols are always followed.

In this paper, we propose a secure e-commerce transaction model. We focus our attention on modeling the security aspects at the transaction level, and discuss the properties an e-commerce transaction protocol must have in order to support the semantics of the transactions. In our environment the players are not trustworthy, so the protocol may not always be followed faithfully. The semantics we use is based in the first instance on the notion of atomicity proposed in [13], and then extended to a broader context where the behaviors of untrustworthy players are taken into consideration.

The rest of the paper is organized as follows. In Section 2, we describe the structure of secure e-commerce transactions, the properties they have and transaction protocols. In Section 3, we concentrate on the issue of how to maintain atomicity with untrustworthy players. In Section 4, we present some examples to show how the idea presented in the previous sections works. We conclude the paper by presenting the related work and summarizing the main results.

2. E-COMMERCE TRANSACTIONS

2.1 Structure

An e-commerce transaction is carried out by *players*. Each player plays a role in the transaction. We always include two players, a customer and a merchant. The role of the customer is to make a payment to the merchant and as a result receives some goods. The role of the merchant is to deliver the goods in exchange for the payment he receives from the customer. There is another special player, termed *arbiter*. An arbiter does not participate in the normal execution. Its only role is to resolve a dispute if one is raised. In reality, there may also be other players involved in a transaction, such as banks, which are responsible for the real transfer of funds which realize the payment between the customer and the merchant, intermediate servers that perform certain authentication for the players, etc. Some players are considered trustworthy while the others are not. A player is trustworthy if it never makes false statement and never behaves inconsistently with the predefined rules during the transaction execution. In general the trustworthiness of players depends on the context. (We assume the arbiter is always trustworthy.)

An e-commerce transaction is a sequence of three logical operations: *negotiation*, *purchasing*, and an optional *remediation*. During the process of negotiation, customers and merchants try to reach an agreement on the terms and conditions on the goods to be purchased and their prices. A successful negotiation results in an order placement in terms of an order form. The order form legally binds both the customer and the merchant. Purchasing operation consists of two sub-operations, *fund transfer* and *goods delivery*, which may or may not be ordered. In fund transfer operation the customer makes payment to the merchant. In goods delivery operation the merchant delivers the goods to the customer. The goods referred to here can be either electronic goods such as software, electronic documents, information services, etc., or tangible goods. The operation remediation refers to a process to resolve disputes that may subsequently arise following the purchasing operation. We will discuss remediation in more detail in Section 3.

2.2. Capabilities

Conceptually, an e-commerce transaction is secure if players' interests are protected from both external attacks and malfunctions of internal components. The former is dealt with by secure messages [10, 12]. The latter is handled by integrity constraints. In this paper, we consider only the latter. The following constraint, *money atomicity*, is proposed in [13]:

Definition 1: A fund transfer operation preserves *money atomicity* if once the customer makes a payment the merchant will receive it and vice versa.

For goods delivery operation, we have similar constraint.

Definition 2: A goods delivery operation preserves *goods atomicity* if once the merchant dispatches the goods the customer will receive it and vice versa.

Definition 2 is different from goods atomicity defined in [13] since it has been separated from money atomicity. Such a separation makes it easier to model different kinds of anomalies. (Refer to the discussion below.)

Definition 3: An e-commerce transaction preserves *purchasing atomicity* if (1) fund transfer preserves money atomicity, (2) the goods delivery operation preserves goods atomicity and (3) either the order has been placed and the customer has paid according to the order and she gets the goods with the value specified on the order, or none of these three things has (effectively) occurred.

In reality, money atomicity is widely realized using infrastructures that perform debiting and crediting in an atomic action. Also, the goods atomicity is realized by infrastructures that ensure dispatching and receiving goods are done in an atomic action. In the subsequent discussions, we will assume the existence of these infrastructures. Thus to ensure purchasing atomicity all we need is to ensure the third condition. (The value of the goods includes such things as their quantities, their physical conditions, and the correctness of their types.)

While ensuring the above condition may not seem to be a difficult problem in normal case, it is more complicated if both customers and merchants may be not trusted, in addition to the possible hardware failures. For example, if the customer has made the payment but the fraudulent merchant did not deliver the goods, then the purchasing atomicity is broken. As another example, suppose a customer makes a payment and the merchant delivers the goods. But later on the customer claims she did not receive it while in reality she did. If somehow she won the case and the merchant had to deliver the goods once more, the atomicity was (effectively) destroyed, too, since in the end the customer received the goods with twice the value she had paid for. In order to cope with this kind of problems, we must identify the causes for the violations of the atomicity. We first introduce some concepts.

2.3. States and Actions

In some cases, it is more convenient to describe atomicity in terms of *states* of an e-commerce transaction. We simply view a state in an e-commerce transaction as an assignment of values to a collection of abstract data items *order*, *money* and *goods*. Data item *order* takes value one if an order has been placed, and zero otherwise. We add to data item *money* an amount transferred if fund transfer is completed, and zero otherwise. We store into data item *goods* the quantity of the goods that are delivered, and zero otherwise. (Think of data item *goods* as a warehouse where one item will never 'over write' another item.)

Actions are the physical operations that implement the logical operations in an e-commerce transaction. An action is a proposition, which claims the occurrence of that action. In most cases, the proposition involves a single message delivery. A proposition may also claim non-occurrence of an action. For example, 'Bank did not send confirmation to merchant' claims non-occurrence of action 'Bank sent

confirmation to merchant'. Clearly, the truth value of a proposition depends on the transaction state. For example, the above proposition is true at the initial state, but false after the bank has sent confirmation to the merchant.

Each action has an *initiator*, which is the subject part in the action statement. In particular, if an action is a message delivery, the sender of the message is always the initiator. In the following, we will use $\text{initiator}(a)$ to denote the initiator of action a .

At the low level an e-commerce transaction is just a sequence of actions. The exact set of actions and their ordering depend on how the logical operations of the transaction are implemented, i.e., the *transaction protocol*. Actions can cause state transitions. Let s be a sequence of actions, and Q and R be two states. We use the notation $Q \rightarrow_s R$ to denote that s causes a state transition from Q to R . More precisely, the transaction is in state Q before any action in sequence s occurs. After all the actions in s have occurred, the transaction enters state R . We say that state R preserves atomicity if $Q \rightarrow_s R$ where Q is the initial state and the occurrences of the actions in sequence s preserve purchasing atomicity.

2.4. Protocols for Secure E-Commerce Transactions

An e-commerce transaction protocol is a collection of rules that governs how the operations in an e-commerce transaction should be executed. Among other things, it specifies the exact set of actions, and how they (or some of them) are ordered to realize purchasing atomicity under various circumstances. It however does not tell how to handle exceptions arising from behaviors of untrustworthy players that may compromise the atomicity. A protocol can be conveniently described by a tree, called *protocol tree*. Shown in Figure 1 is a protocol tree for the protocol in Section 5.1. Each path originating from the root represents a possible sequence of actions executions, which is termed a *run* of the protocol. Thus a run is a class of transaction executions that follow the same execution pattern. The nodes of the tree represent the stages of possible runs of the protocol. A node has a *content*, which is the transaction state at that stage. The letters inside the nodes identify the nodes and their subscripts distinguish contents. Two nodes have the same contents if and only if the inside letters have the same subscripts. The arcs, and therefore paths in general, are state transitions. Each arc is labeled with the action that causes the transition. A path originating from the root and ending at a leaf is called a *complete path*. A run may or may not be protocol conformance. We require that all the runs conforming to the protocol be presented in the tree. If a run does not follow the protocol but can be predicted in advance, then it can also be represented. For example, in Figure 1 any path originating from the root with all its nodes marked with a letter G represents a run that follows the protocol, and any path starting from the root with at least one node marked with a letter B represents a run that does not follow the protocol. If a complete path represents a run that follows the protocol, then it will terminate at a state that preserves atomicity, otherwise it will not. The latter case happens if some players do not respect the protocol. States which do not preserve atomicity will most likely cause disputes. In practice, however, disputes can occur at any node the e-commerce transaction currently proceeds to.

3. REMEDIATION REVISITED

3.1. Atomicity-Aware Disputes

The main function of remediation is to handle disputes. In this process it tries to prevent purchasing atomicity from being destroyed to the best possible extent. A dispute may be *atomicity-aware* or *atomicity-unaware*. An atomicity-unaware dispute will not affect atomicity¹. In the subsequent discussions we consider only atomicity-aware dispute.

3.2. Operational Aspects

Remediation is a collection of zero or more operations, called *dispute resolution*. Each dispute resolution consists of three parts, *dispute initiation*, *investigation* and *decision making*.

3.2.1. Dispute Initiation

A player may initiate a dispute because he/she feels that he/she has been treated unfairly. We term the player who initiates a dispute the initiator of the dispute. The initiator of a dispute submits a *complain* and a *request*. The complain is simply a statement. If the dispute is raised due to the violation of atomicity, the complain statement will involve the occurrence and/or non-occurrence of some actions performed or should have been performed during the execution of the transaction. The request contains some actions that the initiator wishes to be taken.

3.2.2. Investigation

The goal of this operation is to let the arbiter *construct* the current transaction state. More precisely, let $Q \rightarrow R$ be a state transition where Q is the initial state and R an arbitrary state. Suppose a dispute arises when the transaction is in state R . The arbiter can construct state R if he knows sequence s . If the arbiter can indeed construct R , it will be clear whether or not the complain statement is true and whether or not the initiator's request is acceptable. Unfortunately, constructing a complete current state is not always feasible. This is because all the required information may not be recorded. Most importantly, our assumption is that some players are not trustworthy. Thus, all the players may not be fully cooperative. As a result, a player may withhold the evidence for an action he/she has taken, or may claim he/she has taken an action but the evidence is lost while in reality he/she did not take that action. (An analog from criminology field would be that a criminal is rarely cooperative in providing evidence to show that he/she has committed the crime.) If any of these happens, it would be difficult, if not impossible, to know the exact sequence s . The issue here is to maximize the likelihood that this is possible.

¹ An example of atomicity-unaware dispute is "The customer had a personal attack on me, I demand her apology".

First observe that not all the actions change the transaction state. For example, during the negotiation a customer may send a message to the merchant to request for information about the products. Before the fund transfer actually happens, the customer sends her (encrypted) account information to the merchant, etc. An action that alters the transaction state is said to be *atomicity-sensitive*. Clearly, only atomicity-sensitive actions are necessary in constructing the transaction state. Second, for any player, there exist actions such that if he/she has taken those actions, admitting this fact will not conflict with his/her best interest in any dispute. Similarly, for any player there exist actions which he/she should be willing to prove not to have been taken by other players. For example, if a customer has made a payment, there is no reason why she will deny that fact. On the other hand, if she did not receive the goods, it is unlikely she will say the opposite. To make the idea more formal, we use the notion of *no-conflict condition*. A no-conflict condition contains two parts, *no-conflict on occurrence* and *no-conflict on non-occurrence*, which are represented respectively by predicates $\text{no-conflict}_o(p, a)$ and $\text{no-conflict}_n(p, a)$ where p is a player and a is an action. A no-conflict condition should be defined in such a way that $\text{no-conflict}_o(p, a) = \text{true}$ if and only if player p admitting occurrence of action a does not conflict with p 's best interest, and $\text{no-conflict}_n(p, a) = \text{true}$ if and only if player p admitting non-occurrence of action a does not conflict with p 's best interest. (Note that the term 'admitting' must not be omitted. For example, stealing probably does not conflict with the interest of the theft, but admitting it definitely is.) Clearly, in reality whether or not admitting occurrence/non-occurrence of any action conflicts with the interest of a player is subject to interpretation. It is therefore up to designers to decide precisely how these predicates should be defined.

Definition 4: Let $\text{no-conflict}_o(p, a)$ and $\text{no-conflict}_n(p, a)$ be a no-conflict condition. For any player p , its *benefit set*², denoted as $\text{benefit}(p)$, for the given no-conflict condition is a set of propositions such that for all $q \in \text{benefit}(p)$ (1) q claims either occurrence or non-occurrence of some action and (2) if q claims occurrence of action a then $\text{no-conflict}_o(p, a) = \text{true}$ and if q claims non-occurrence of action a then $\text{no-conflict}_n(p, a) = \text{true}$.

Note that the constraints described here are necessary only for untrustworthy players. If a player is trusted, however, those constraints are not needed. The above definition considers players' cooperation only. As mentioned previously, however, the ultimate goal of players' cooperation is to construct the complete state of the transaction in a dispute. Thus we must impose further requirements on benefit sets.

Definition 5: Given a no-conflict condition as in Definition 4, a set of benefit sets for that condition is *state constructive* if for each atomicity sensitive action, its occurrence is claimed by some proposition in some benefit set and its non-occurrence is also claimed by some proposition in some benefit set.

The key property of a state constructive set is described in the following theorem. We now list the notations that will be used: $p[a]$ and $p[\neg a]$ denote propositions that

² The intuition for this naming is that an arbiter can 'benefit' from the set by having the player's cooperation.

claim occurrence and non-occurrence, respectively, of action a ; for a set B of benefit sets, $\beta(B)$ denotes the union of all the benefit sets in B ; for any proposition p and state X , $p(X)$ denotes the value of p when transaction is in state X .

Theorem 1: Given a no-conflict condition as that in Definition 4, and a state constructive set B of benefit sets for that condition. Let $N \rightarrow_s R$ be a state transition where N is the initial state and R an arbitrary state. Then for any atomicity sensitive action a , $a \in s$ if and only if $\forall p[a] \in \beta(B), p[a](R) = \text{true}$.

Theorem 1 states that when transaction is at state R , for all the propositions in the benefit sets which evaluate to true and which claim occurrences of some actions, the actions they claim constitute a complete set of atomicity sensitive actions in sequence s . Thus if we know how these actions are ordered in s , we can construct state R . (Knowing the ordering of actions is not a problem since this is determined by the protocol used, which is given in advance.)

During the investigation, the arbiter determines the trustworthiness of each player, and constructs the associated benefit set accordingly such that the set of all the benefit sets is state constructive. (We defer the discussion on how the benefit sets are actually constructed to Section 3.3.1.) The arbiter then requests the players to prove the propositions in their benefit sets. Thus, hopefully, all the players will be cooperative. This will allow the arbiter to construct a complete state and come up with a decision at the next decision making step.

3.2.3. Decision Making

This is where the arbiter decides if the request by the dispute initiator will be honored or rejected. The arbiter bases his decision on the state he can construct and uses purchasing atomicity as the criterion as much as he can. If every player can establish the truth values for all the propositions that take truth value of true in the associated benefit set when the dispute arises, the arbiter can construct the complete transaction state. In this case a decision can possibly be made algorithmically, otherwise the arbiter has to exercise his discretion by considering whatever other factors he feels are necessary to assist him in making a decision. This includes, for example, any documents the players can present, the trustworthiness he has on each player, etc.

3.3. Implementing Arbiter As A Two Tier Structure

Clearly an arbiter plays a dominant role in dispute resolution. Its decision should be consistent and the process should be efficient. To attain this goal, it will be appropriate to implement an arbiter using a two tier structure, a computer software and a human, which are henceforth called software arbiter and human arbiter respectively. The software arbiter generates algorithmic solutions. These include constructing benefit sets and in certain cases resolving disputes. The human arbiter does the processing that requires human judgment.

3.3.1. Constructing Benefit Sets

With a protocol tree, atomicity-sensitive actions are simply those arcs that are incident to the nodes with different contents. In terms of the notations we introduced in Section 2.4, these are the arcs whose two end nodes contain the inside letters with different subscripts. To determine benefit sets, we must have a priori no-conflict condition. As mentioned before, the exact definition of this condition depends on a designer's interpretation of 'conflicts' in his context. We now propose a definition, called CAI condition (Contribute to Atomicity Installation). In the following expression free variable s denotes a sequence of actions. For any X , X_{prev} denotes a state that preserves atomicity. We use N for the initial state.

Definition 6 (CAI condition): For any action a and player p :

$$\begin{aligned} \text{no-conflict}_o(p, a) &\equiv (p = \text{initiator}(a)) \ \& \ \exists s \ X_{prev}, [N \rightarrow_s X_{prev} \ \& \ a \in s] \quad \text{I} \\ &\quad \text{or } (p \neq \text{initiator}(a)) \ \& \ \neg \exists s \ X_{prev}, [N \rightarrow_s X_{prev} \ \& \ a \in s] \quad \text{II} \\ \text{no-conflict}_n(p, a) &\equiv (p = \text{initiator}(a)) \ \& \ \neg \exists s \ X_{prev}, [N \rightarrow_s X_{prev} \ \& \ a \in s] \quad \text{III} \\ &\quad \text{or } (p \neq \text{initiator}(a)) \ \& \ \exists s \ X_{prev}, [N \rightarrow_s X_{prev} \ \& \ a \in s] \quad \text{IV} \end{aligned}$$

Condition I states that the initiator of action a is p and action a contributes to an installation of atomicity. Condition II states that the initiator of action a is not p and action a does not contribute to any installation of atomicity. Conditions III and IV can be interpreted similarly. Our philosophy is the following. An action that contributes to installing an atomicity is 'good', therefore its initiator probably will be willing to admit that he/she has done it and other player may have less reservation to prove that the initiator has not done it. On the other hand, if an action contributes to destroying the atomicity, then it is 'bad', and we interchange the subjects in the above two sentences. Note that sequence s following symbol ' $\exists$ ' is represented by a complete path terminating at a node that preserves atomicity in protocol tree. As a result, CAI condition is easily identified in the protocol tree. The following algorithm constructs the benefit set for each player. For player X , we use $P(X)$ to denote the supposed benefit set for him/her.

Algorithm 1 – constructing benefit sets

1. Construct the protocol tree for the transaction protocol;
2. Mark each action whose incident nodes have the same contents as atomicity insensitive, and the remaining actions as atomicity sensitive.
3. For each complete path that terminates at a state that preserves atomicity, mark every node in it with letter G; \ sequences that meet $\exists$ condition in Definition 6
4. Mark every node that has not been marked at step 3 with letter B; \ those that meet $\neg \exists$ condition
5. For each atomicity sensitive action A which ends at a node marked with G, include into set $P(\text{initiator}(A))$ a statement claiming occurrence of A . If $\text{initiator}(A)$ is a trusted player, include into set $P(\text{initiator}(A))$ a statement claiming non-occurrence of A , otherwise include into set $P(J)$ a statement

claiming non-occurrence of A where $J \neq \text{initiator}(A)$; \\\by I and IV for not-trusted players

6. For each remaining atomicity sensitive action A, include into set $P(\text{initiator}(A))$ a statement claiming non-occurrence of A. If $\text{initiator}(A)$ is a trusted player, include into $P(\text{initiator}(A))$ a statement claiming occurrence of A, otherwise include into set $P(J)$ a statement claiming occurrence of A where $J \neq \text{initiator}(A)$. \\\by III and II for not-trusted players

The correctness of this algorithm is established by the following theorem.

Theorem 2: Let $S \equiv \{P(J): P(J) \text{ is constructed by Algorithm 1}\}$. Then S is a state constructive set of benefit sets for the CAI condition.

3.3.2. Generating a Decision

In this process, the software arbiter first tries to make a decision. If this is not possible, it will turn the case over to the human arbiter. The following algorithm is executed by the software arbiter. As mentioned in Section 3.1, we consider atomicity-aware disputes only. Thus we assume atomicity-unaware disputes will not be submitted.

Algorithm 2 — decision making

1. if truth values of all propositions that take truth value of true in benefit sets are obtained then construct complete transaction state
2. if the state preserves the atomicity then
3. if the initiator can prove the complain statement then
4. suggestion: consider the reasonableness of the compensation request
5. else // the initiator cannot prove his/her complain
6. no action is taken
7. else // the state does not preserve the atomicity
8. if the request by the initiator reinstalls atomicity then
9. accept the request
10. else // the request does not reinstall atomicity
11. if the initiator can prove the complain statement then
12. suggestion: consider the reasonableness of the compensation request
13. else // the initiator cannot prove his/her complain
14. no action is taken
15. else // only partial state can be constructed
16. human judgment

In the above algorithm a line containing suggestion indicates that the software arbiter does not make a final decision but only gives a suggestion to the human

arbiter to assist in his/her making a decision. Note that at line 4, we are sure that the initiator of the dispute has complained something unrelated to atomicity. Thus the request will destroy atomicity, since otherwise the dispute is not atomicity--aware, which we have assumed will not be submitted. This means that the initiator has requested for compensation. It is therefore necessary to use the human judgment to decide if the request is reasonable. If the current state does not preserve atomicity but the initiator's request reinstalls it, we immediately grant the acceptance to the request, as described at line 9. This is because atomicity is our correctness criterion and any request that ends up with preserving it should be considered justifiable. In this case there is no need to consider if the initiator can prove his/her complain. Note that we do not consider the kind of requests that reinstall atomicity but at the same time demand something else, since this extra demand must be unrelated to atomicity, which by our assumption will not be handled by the algorithm.

4. EXAMPLE

In this section, we look at an example of e-commerce transactions. We show how the concepts in the previous sections are realized in terms of the example. In the protocol, we omit the specification of the message content. We assume that all three conditions for a secure message are met. Therefore we describe the protocol only in terms of pseudo language. In this example, we assume the bank is considered by the arbiter as a trusted player while the customer and the merchant are not.

5.1. Protocol Specification

The normal execution of the protocol involves three players, customer, merchant and bank. The following is how they interact.

Protocol³

1. The customer sends an order form to the merchant, which includes the specification of the goods, price, and her credit account information;
2. The merchant checks if the information on the order form is acceptable. If it is, he sends confirmation to the customer, otherwise, sends negative confirmation to the customer and stop;
3. The merchant checks the total price of the goods ordered. If it does not exceed a threshold, deliver the goods immediately using the goods delivery infrastructure and then sends to the customer a signed multi-copy receipt which uniquely identifies the goods delivered;
4. The merchant submits the account information to the bank for fund transfer;

³ This is a modified version of the high level specification of NetBill protocol proposed in [13].

5. If everything relating to fund transfer is valid, the bank performs fund transfer and sends a confirmation to the merchant, otherwise sends a rejection to the merchant;
6. If the merchant receives from the bank a confirmation for fund transfer, then if he has not done so he delivers the goods using the goods delivery infrastructure and sends to the customer a signed multi-copy receipt that uniquely identifies the goods delivered;
7. If the merchant receives from the bank a rejection for fund transfer and he has not delivered the goods, cancel the order and sends to the customer an order cancellation notification;
8. After she receives the goods the customer signs all the copies of the receipts, keeps one copy of it, and returns the rest to the merchant.

Steps 1 and 2 constitute the negotiation, while purchasing operation is implemented by the remaining steps. Note that the protocol does not mention what would happen if the merchant had received from the bank a rejection for fund transfer but he had already delivered the goods. Since this is not specified by the protocol, it can be resolved only by raising a dispute. (Presumably, the merchant does not mind the hassle and risk of losing money in exchange for a better customer satisfaction in case the total value of the goods is not very big.) We assume that once receipt has been sent, the recipient will get it. The protocol tree is depicted in Figure 1.

4.2. Scenarios

We now look at how the transaction semantics is implemented under several scenarios. In this protocol tree, some runs which do not follow the protocol have also been represented. These are the paths originating from the root in which some nodes are marked with B. These nodes will most likely to trigger disputes. We use 'merchant deliver good' to indicate that the goods delivered by the merchant match in value the payment by the customer, and 'merchant deliver bad' to denote that the goods are of less values than the payment. The latter happens, for example, when the goods are damaged, wrong kind, etc. In this case, we assume it is always possible for the customer to show that the goods are bad. (Refer to Section 2.4 for the meaning of other notations.) By following the steps in Algorithm 1, we first identify atomicity sensitive actions (those without an asterisk) and then mark each node properly as shown in the figure. Then the following benefit sets are generated:

P(CUSTOMER):

- C1: MERCHANT did not confirm the order
- C2: MERCHANT did not cancel order
- C3: MERCHANT did not deliver good goods
- C4: MERCHANT delivered bad goods

P(MERCHANT)

- M1: MERCHANT confirmed the order
- M2: MERCHANT canceled the order
- M3: MERCHANT delivered good goods
- M4: MERCHANT did not deliver bad goods

P(BANK)

- B1: BANK confirmed fund transfer from the customer to the merchant
- B2: BANK did not confirm fund transfer

Scenario 1: When the transaction reaches node R_6 , the customer initiates a dispute complaining 'The merchant delivered bad goods to me' and requesting 'exchange for good'. The software arbiter, after constructing the above benefit sets, requests the players to prove any propositions in the benefit sets that are true at node R_6 . There are five propositions that are true at that node, C2, C3, C4, M1 and B1. The customer proves C2 by showing the signed order form. She proves C4, and therefore C3, by showing that the goods identified on the signed receipt are indeed bad. The merchant proves M1 by showing the signed order form. The bank proves B1 by showing the fund transfer record. (Since the bank is trustworthy, any document it presents is considered legally acceptable.) Now the software arbiter has constructed the complete transaction state at node R_6 . This state does not preserve atomicity, nonetheless the customer's request reinstalls it. Thus line 9 in Algorithm 2 is reached. The request is honored.

Scenario 2: The transaction reaches node S_5 . The customer initiates a dispute with complaint 'The merchant promised me a 25% discount if I placed order before June 12, but he did not keep his promise after I sent the order in June 11' and request '

refund 25% of the amount I have paid '. The software arbiter again constructs the above benefit sets and requests the players to prove propositions that are true. The following propositions are true at node S_5 : C2, M1, M3, M4 and B1. Again, all these propositions are proven by the players respectively. Now the arbiter requests the customer to prove her complain statement. (Suppose the merchant's promise is not written on the order.) If she can, line 4 in Algorithm 2 is executed, otherwise line 6 is executed. Note that in the former case, the customer is actually requesting for a compensation. The software arbiter is unable to determine if such a request is reasonable. It therefore passes the case over to the human arbiter.

Scenario 3: The transaction reaches node M_5 . Before the customer returns the receipt to the merchant, she initiates a dispute by making a false claim ' I did not receive the goods' and requesting ' The merchant refund my payment'. The propositions that are true at this node are C2, M1, M3, M4 and B1, where C2, M1 and B1 can be proven by the corresponding players. However, the merchant is unable to prove M3 (and M4), since he does not have the receipt from the customer. Thus the software arbiter is unable to construct a complete transaction state at node M_5 . (More precisely, he does not know if the merchant indeed delivered goods or not.) A human judgment is needed to resolve the case.

5. RELATED WORK AND CONCLUDING REMARKS

E-commerce transactions have been the targets for an extensive study in the last decades. A comprehensive review of the current states and the challenges are given in [7]. Many protocols with varying levels of security guarantees have been proposed [2, 3, 4, 5, 8, 12 16]. Atomicity is an issue that has been thoroughly explored in database transactions in the last two decades [6, 9]. Its attachment to e-commerce transactions is introduced in [13 14]. In [11 15] the authors give a more detailed analysis on the transactional properties of e-commerce transactions. In [1], the authors propose a framework for dispute handling in e-commerce transactions. In [12], the authors describe several properties of secure electronic transactions.

An e-commerce transaction is a very different kind of transaction from any of the traditional computerized operations. Due to the potential hostile execution environment the requirement for its secure execution is more difficult to ensure than that for traditional transactions. This brings about the issue of implementing the transaction semantics not only in the face of machine related anomalies, but also the exceptions resulting from human behaviors. In this paper, we study a secure e-commerce transaction model in order for a better understanding of its structures and behaviors. The starting point for the semantics of our model is atomicity proposed elsewhere. We then extend it to a broader context and study various problems and approaches to attacking these problems in its realization. We also study a property for a protocol to provide support to realizing atomicity. Finally we present some related issues that deserve further study.

REFERENCES

1. N. Asokan, E. Herrweghen and M. Steiner, "Towards a Framework for Handling Disputes in Payment Systems", Proc. of 3rd USENIX Workshop on Electronic Commerce, 1998.
2. M. Bellare, J. Garay, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik and M. Waidner, "iKP – a Family of Secure Electronic Payment Protocols", Proc. of the 1st USENIX Workshop on Electronic Commerce, 1995.
3. D. Chaum, "Achieving Electronic Privacy", Scientific American, 1992, pp 96 – 101.
4. B. Cox, J. D. Tygar and M. Sirbu, "NetBill Security and Transaction Protocol", Proc. of the 1st USENIX Workshop on Electronic Commerce, 1995.
5. E. Gabber and A. Silberschatz, "Agora: a Minimal Distribut Protocol for Electronic Commerce", Proc. of the 2nd USENIX Workshop on Electronic Commerce, 1996.
6. J. Gray and A. Reuter, " Transactions Processing: Techniques and Concepts", Morgan Kaufmann, San Mateo, CA, 1994.
7. S. Kimbrough and R. Lee, "Formal Aspects of Electronic Commerce: Research Issues and Challenges", International Journal of Electronic Commerce, Vol. 1, No. 4, 1997, pp 11 – 30.
8. J. H. Lee, "A Resilient Access Control Scheme for Secure Electronic transactions", Proc. of the 2nd USENIX Workshop on Electronic Commerce, 1996.
9. N. Lynch, M. Merritt, W. Weihl and A. Fekete, "Atomic Transactions", Morgan Kaufmann, San Mateo, CA, 1994.
10. R. Rivest, A. Shamir and L. Adleman, " Method for Obtaining Digital Signatures and Public-Key Cryptosystems", CACM 21 (2), 1978, pp. 120 -126.
11. H. Schuldt, A. Popovici and H. J. Schek, " Execution Guarantees in Electronic Commerce", Proc. of 8th Intl Workshop on Foundations of Models and Languages for Data and Objects In: Gunter Saake, Kerstin Schwarz, Can Türker (eds.), Transactions and Database Dynamics. Lecture Notes in Computer Science Nr. 1773, Springer Verlag, Berlin, December 1999, pp. 193 –202.
12. D. Steves, C. Yurkanan and M. Gouda, "A Protocol for Secure Transactions", Proceedings of the 2nd USENIX Workshop on Electronic Commerce (EC 96) Oakland, California, November 18 - 21, 1996. http://www.sage.usenix.org/publications/library/proceedings/ec96/full_papers/steves/html/index.html.
13. J. D. Tygar, " Atomicity in Electronic Commerce", PODC 96, pp 8 – 26.
14. J. D. Tygar, " Atomicity versus Anonymity : distributed Transaction Electronic Commerce", Proc. of the 24th VLDB Conference, 1998, 1 – 12.
15. J. Veijalainen, "Transactions in Mobile Electronic Commerce", Proc. of 8th Intl. Workshop on Foundations of Models and Languages for Data and Objects. In: Gunter Saake, Kerstin Schwarz, Can Türker (eds.), Transactions and Database Dynamics. Lecture Notes in Computer Science Nr. 1773, Springer Verlag, Berlin, December 1999, pp. 208- 229.
16. VISA, Mastercard, et.al, "Secure Electronic Transaction Specification", book 2, Technical Specification, [http://www.visa.com/cgi-bin/vee/sf/set/settech.html?](http://www.visa.com/cgi-bin/vee/sf/set/settech.html?2+1) 2+1, 1996.