Preserving Anonymity in Fair Exchange Complex Transactions E-Commerce Protocol for B2C/B2B Applications

Cătălin V. Bîrjoveanu[1] and Mirela Bîrjoveanu[2]
[1]*Department of Computer Science, "Al.I.Cuza" University of Iaşi, Iaşi, Romania*
[2]*Continental Automotive, Iaşi, Romania*

Keywords: Electronic Commerce Security, Complex Transactions, Anonymity, Fair Exchange, Security Protocols.

Abstract: In the electronic commerce context, it is common that a customer wishes to buy a pack of products/services composed of several products/services from different merchants. The customer is interested in buying all products from the pack or no product at all. On the other hand, sometimes the customer wants to buy exactly one product from many merchants, and for this, he specifies in his request more possible products according to his preferences but from these options only one will be committed. The combination in any form of these two types of e-commerce transactions will be named complex transaction. Despite the great variety of multi-party fair exchange protocols proposed until now, there is no solution to address to physical products delivery in complex transactions. In this paper, we propose the first e-commerce protocol for physical products delivery in complex transactions that provides fair exchange and anonymity of the customer.

1 INTRODUCTION

In the electronic commerce, there are situations in that a customer wishes to buy a pack of products/services composed of several products (physical or digital)/services from different merchants. In this type of e-commerce transactions, the customer is interested in buying all products from the pack or no product at all, namely *aggregate/atomic transactions*. For flexibility, in the *optional transactions*, the customer wants to buy exactly one product from many merchants, and for this, he specifies in his request more possible products according to his preferences but from these options only one will be committed. We will refer to the combination in any form of aggregate and optional transactions as *complex transactions*.

In e-commerce protocols, *fair exchange* is an essential property. For complex transactions, an e-commerce protocol in that a customer wishes to buy many products from different merchants assures fair exchange if:

- for any optional transaction from the complex transaction, the customer obtains exactly one product in order of his specified preferences, and
- for any aggregate transaction from the complex transaction, the customer obtains all products,

and each merchant obtains the payment for the corresponding product, or none of them obtains nothing. In a complex transaction, the issues that can appear are when in a aggregate transaction some products can be successfully acquired, but the others not, or when in an optional transaction more than one product is successfully acquired. In these cases, even if each corresponding merchant gets the payment for his product, fair exchange is not ensured.

To achieve fair exchange, most of the proposed protocols are based on a Trusted Third Party (TTP) that acts like item validator or to solve the disputes.

Anonymity of the customer is a key property that must be considered in a fair exchange e-commerce protocol. The customer may not want to reveal sensitive data of his identity (as credit card number, information about customer's bank, customer's account number) so that this information can not be used by merchant in commercial purpose to build spending habits of the customer. An e-commerce protocol provides the customer's anonymity if no party and no coalition between parties can link the customer's true identity with his actions. To obtain customer's anonymity in our protocol,the challenge is to guarantee this requirement both in payment and collection of physical products.

Until now there are protocols proposed for physical products delivery that provide fair exchange and

Bîrjoveanu, C. and Bîrjoveanu, M.
Preserving Anonymity in Fair Exchange Complex Transactions E-Commerce Protocol for B2C/B2B Applications.
DOI: 10.5220/0006850800990110
In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) - Volume 1: DCNET, ICE-B, OPTICS, SIGMAP and WINSYS*, pages 99-110
ISBN: 978-989-758-319-3

consider only one customer and one merchant (Birjoveanu, 2015; Djuric and Gasevic, 2015; Alaraj, 2012; Li et al., 2006; Zhang et al., 2006) and from all this solutions the one proposed in (Birjoveanu, 2015) provides also anonymity for both customer and merchant.

There are known many multi-party fair exchange protocols proposed with applications in e-commerce transactions for buying digital goods (Liu, 2009), digital signature of contracts (Draper-Gil et al., 2013; Onieva et al., 2009), and certified e-mail (Onieva et al., 2009). Despite great variety of multi-party fair exchange protocols proposed until now, there is no solution to address our problem: complex transactions where a customer wants to buy several physical products from different merchants, providing fair exchange and anonymity.

From all known solutions for multi-party fair exchange, no one can be applied to our problem. First, all proposed solutions exchange digital items, while our problem involves exchange between electronic payment and physical products delivery. Secondly, some multi-party non-repudiation solutions exchange different messages (Onieva et al., 2009) in a one-to-many configuration, without taken in to consideration atomicity, while our problem involves one customer that wants to buy several products while preserving atomicity. Thirdly, the scenario most closed to our scenario is the one proposed by (Liu, 2009), where in an aggregate transaction, a customer wants to buy from different merchants several digital products. The solution from (Liu, 2009) can not be applied to our problem because the protocol architecture for physical products delivery is more complex than for digital products: delivery agents are needed for physical products delivery from merchants to customer. Also, in (Liu, 2009) optional transactions and anonymity are not considered.

Our contribution. In this paper, we propose a new anonymous fair exchange e-commerce protocol for complex transactions in that the customer wants to buy several different physical products from different merchants. In our scenario, a complex transaction is a combination in any form of aggregate and optional transactions. Our solution is the first that addresses to physical product delivery in complex transactions. Also, our protocol provides non-repudiation, integrity and confidentiality of data exchanged between the parties.

The paper is structured as follows: section 2 gives an informal description, section 3 presents the protocol, section 4 provides an analysis of the proposed protocol and section 5 contains the conclusion.

2 INFORMAL DESCRIPTION

Our protocol has applications in Business to Consumer (B2C) and Business to Business (B2B) scenarios. For a B2B scenario, the customer is the Electron company that manufactures electronic boards for different purposes, on request from his clients. To plan its business, Electron uses an online catalog from where it can buy several electronic components from different merchants denoted by $M1, M2, M3$, e.t.c. From the online catalog, Electron can select products like: resistors (R), capacitors (C), integrated circuits (IC), cables, connectors, printed circuit boards (PCB) and so on. Electron wants to start the production of a new electronic board and therefore wants to prepare its order in form of an e-commerce complex transaction as follows: (100R of $10k\Omega$ from $M1$ or 70R of $20k\Omega$ from $M2$) and (50C of 100mF from $M3$ or 100C of 70mF from $M4$) and 70 connectors $DB35$ type from $M5$ and 30PCB from the $M6$. The complex transaction is composed from an aggregate transaction and two optional transactions. For the first optional transaction if Electron can not acquire 100R of $10k\Omega$ from $M1$ due to lack of stock or delay in delivery time, then its second option is taken into consideration to acquire 70R of $20k\Omega$ from $M2$. To start the production, Electron needs all types of components specified in its request, so a partial combination (e.g. 100R of $10k\Omega$, 100C of 70mF and 30PCB, but without 70 connectors $DB35$ type) is not useful for him. For an optional transaction, Electron must not acquire more than one product (e.g. for the first optional transaction he must not acquire both 100R of $10k\Omega$ and 70R of $20k\Omega$) because then he will remain with unnecessary products. For example, a pack of products that solves the customer's options is: 100R of $10k\Omega$, and 100C of 70mF, and 70 connectors $DB35$ type and 30PCB.

A similar scenario can be used in B2C applications. In this case, the customer is a person that likes electronics and wants to build an electronic hobby kit, and for this he uses the online catalog to order the needed components.

The protocol we propose uses an online TTP that will validate the customer's coins and will provide fair exchange if any party misbehaves or prematurely aborts. The customer may choose to remain anonymous during the protocol execution. Our protocol uses the electronic cash payment mechanism based on group blind digital signatures on behalf of the banks proposed in (Lysyanskaya and Ramzan, 1998) to provide anonymity of the customer in the payment phase. To ensure anonymity of the customer in the physical delivery phase, our protocol is based on a delivery

agent that takes the product from the merchant and provide it to a destination cabinet, where the access to the physical products is protected by passwords.

Next, we will informally describe the protocol. When the customer decides the products pack he wants to buy and the options for each product from the pack, he clicks a "submit" button on the online catalog. In back-end, the protocol searches a sequence of subtransactions to satisfy the customer's options in order of preferences he supplies. A subtransaction is a sequence of protocol's steps in which the customer buys a certain physical product from a certain merchant.

Our protocol can run in multiple rounds. One protocol round consists of three sub-protocols: the *Agreement* sub-protocol, the *Delivery* sub-protocol and the *Payment* sub-protocol. In *Agreement*, a sequence of subtransactions is started as a possible solution for customer's choices. For each subtransaction, the customer buys a digital coin from his bank and validates it to TTP. The customer sends to the merchant the purchase order and the digital signature of TTP and the merchant replies to confirm the agreement on subtransaction's terms. The *Agreement* will be completed either with all subtransactions successfully finished or all aborted. If all subtransactions successfully finished *Agreement*, then *Delivery* simultaneous realizes the physical delivery of products from these subtransactions. After all products from the subtransactions involved in *Delivery* are posted to the destination cabinet, the customer collects the products and provides it an evidence of the products collection. If *Delivery* is successfully for all subtransactions, then *Payment* simultaneously performs the payment for these subtransactions.

In a protocol round some subtransactions can be aborted without solving the customer's options in that round. If the customer's options are not completely explored, new protocol rounds will be executed to search a sequence of subtransactions that will satisfy his choices. The protocol terminates after a round that solves the customer's options, or after a round where, after *Agreement*, all subtransactions are aborted.

3 THE PROTOCOL

Our proposal uses as a building block our previous work (Birjoveanu, 2015) that involves physical products delivery for only one customer and one merchant and that provides fair exchange and anonymity. The Table 1 presents the notations used in the description of our protocol.

3.1 Protocol Infrastructure

The following assumptions are made for our protocol: (1) All parties use the same algorithms for encryption, hash, digital signature and the same group blind digital signature protocol mentioned in Table 1. (2) Cryptographic algorithms are strong enough. (3) TTP is the group manager, namely Central Bank, that is known by all parties implied in protocol. TTP does not misbehave or collude with any of parties to provide benefits to another party. (4) There are n merchants $M_1, \ldots, M_n$. C can buy a pack of products from these merchants, and each merchant can provide to C only a certain type of product. C and each M_i, have an account to their bank. (5) All banks from group and group manager share a commit-buffer in that each subtransaction's value is stored until the subtransaction is completed successfully or aborted. (6) All banks from group and group manager maintain a global list of coin's serial already spent, validated but unspent, or canceled, to allow any bank to check a coin for double-spending or double-canceling. Each record contains the unique identifier of the subtransaction, the coin's serial and a *spent* flag. The value of *spent* flag corresponds to the current state of the coin: $spent = 0$ means that the coin is validated by TTP but not yet spent, $spent = 1$ the coin has already been spent, $spent = 2$ the coin has already been canceled. (7) For each substransaction s_i, a destination cabinet DC exists, where the physical products are provided by DA_i, and C can collect the product P_i with the identifier Pid_i from DC only by knowing a password that is set by M_i. DC is used to hide the true identity of C if he wants. DC has the ability to digitally sign messages, verify signatures and to check if the password entered by C corresponds to the barcode set on product. After C provides the correct password, DC opens a hatch where packaged product is available to C. DC has a video camera mounted that records when C unwraps each packaged product and check if the product is the ordered one. DC has a device that allows to C, by pushing a button, to send the encrypted recording to TTP. This feature allows on TTP side to store in a buffer *PidsAborted* the product's identifiers that are not received according to the agreement conditions. C uses this feature only if is not satisfied with the product, as an evidence of wrong product reception. Otherwise, the recording is automatically deleted. (8) Communication channels that are set between parties provides anonymity, except the cases in that the parties choose to reveal their true identities.

Table 1: Notations used in the protocol description.

Symbol	Interpretation
C/C'	True identity/pseudo identity of the customer
P_i, DA_i	The product with identifier Pid_i, the delivery agent i, where $1 \leq i \leq n$
M_i	Identity of the merchant i that sells P_i
CB/M_iB	The bank of customer/merchant M_i
C_{acct}/M_{iacct}	The bank account of the customer/merchant M_i with CB/M_iB
Pr_i, Q_i, Po_i	Price, quantity, purchase order used to order P_i
s_i	$(Sid_i, C', M_i, Pid_i, fst)$-the subtransaction in that C' buys P_i from M_i
Sid_i	The unique identifier of s_i
DC_{addr}	The mailing address of the destination cabinet DC
$A \rightarrow B$: m	A sends the message m to B
$DC \Rightarrow C'$: m	DC sends m to DA_i that forwards it to M_i and M_i to C'
A_{pub}, A_{prv}	(RSA) Public/private key pair of A
A'_{ipub}, A'_{iprv}	One time (RSA) public/private key pair of A used only in s_i
$\{m\}_{K'}, h(m)$	m encrypted with K', m's digest obtained by a hash function h (SHA-2)
$sig_A(m)$	(RSA) Digital Signature with the A's private key A_{prv} on $h(m)$
T_{iTTP}, N_{iA}	Timestamp generated by TTP, nonce generated by A in s_i
L_i	Lifetime of encrypted digital coin's validity in s_i
$c_i, K_i, sig_{CB}(c_i)$	Coin generated by C, AES symmetric key used in s_i, CB's signature
	on c_i obtained by Group Blind Digital Signature (GBDS) protocol (Lysyanskaya and Ramzan, 1998)

3.2 Prelude

We assume that before the starting of the protocol, the following system setup steps are executed: (1) TTP generates a public/private key pair, (TTP_{pub}, TTP_{prv}) and provides TTP_{pub} to C and each M_i, where $1 \leq i \leq n$. (2) When C and each M_i create accounts to their banks, each of them generates a public/private key pair, (C_{pub}, C_{prv}) and (M_{ipub}, M_{iprv}), respectively. C provides C_{pub} to CB and each M_i provides M_{ipub} to M_iB. The banks maintain databases with public keys of their clients associated to their accounts. (3) C and each M_i generates a one time public/private key pair (C'_{ipub}, C'_{iprv}), respectively (M'_{ipub}, M'_{iprv}) that each of them will use it only in one subtransaction. (4) The *Setup* and *Join* phases of the GBDS protocol (Lysyanskaya and Ramzan, 1998) are executed. Briefly, the TTP generates a secret key for group manager and the group's public key. CB and each M_iB obtain from TTP the group membership certificate.

3.3 Protocol Description

For an aggregate transaction, we define the *aggregation* operator, denoted by $\wedge$, as follows: $Pid_1 \wedge \ldots \wedge Pid_k$ meaning that C wishes to buy exactly k products with product's identifiers $Pid_1, \ldots, Pid_k$. For an optional transaction, we define the *option* operator, denoted by $\vee$, as follows: $Pid_1 \vee \ldots \vee Pid_k$ meaning that C wishes to buy a product that is exactly one of the products with product's identifiers $Pid_1, \ldots, Pid_k$, where the apparition order of the product's identifiers is the priority given by C. This means that C wishes first of all to buy the product Pid_1, but if this is not possible, his second option is Pid_2, and so on.

From the choices of C describing the sequence of products he wishes to buy, we build a tree over the product identifiers selected by C using $\wedge$ and $\vee$ operators. For efficiency, to represent the tree, we use the *left-child, right-sibling representation* in that each internal node corresponds to one of the above operators or to a product identifier, while each leaf node corresponds to a product identifier. Each node of the tree is represented by a structure with the following fields: *info* for storing the useful information (product identifier or one of the operators), *left* for pointing to the leftmost child of node, and *right* for pointing to the sibling of the node immediately to the right. The access to tree is realized trough the root.

An example of tree derived from the complex transaction from section 2, is shown in Figure 1. Pid_1 corresponds to R of $10k\Omega$, Pid_2 to R of $20k\Omega$, Pid_3 to C of 100mF, Pid_4 to C of 70mF, Pid_5 to connectors $DB35$ type and Pid_6 to PCB. The root node has $\wedge$ operator as *info*. The root does not have any right sibling and its children are the two nodes having $\vee$ operator as *info* and the nodes with the *info* Pid_5 and Pid_6. A parent-child link is realized as follows: the parent

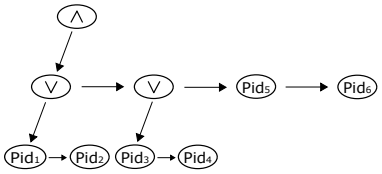

Figure 1: Tree describing the customer's choices in left-child, right-sibling representation.

node points only to its leftmost child, and the rest of its children can be accessed starting with the leftmost child via sibling relationship. The first node having as *info* $\vee$ operator defines the product corresponding to the optional subtransaction $Pid_1 \vee Pid_2$.

Next, we will describe the phases of the protocol.

3.4 Agreement Sub-protocol

We use $s_i.fst$ to denote the flag of a subtransaction $s_i = (Sid_i, C', M_i, Pid_i, fst)$. $s_i.fst$ can have values from the set $\{1, 2, 3, abort\}$. $s_i.fst$ = 1 means that s_i successfully completed *Agreement*, $s_i.fst$ = 2 means that s_i successfully completed *Delivery*, $s_i.fst$ = 3 means that s_i successfully completed *Payment*, and $s_i.fst$ = *abort* means that s_i has not successfully completed *Agreement* or *Delivery*, or successfully completed *Agreement* but had to be later aborted because s_i belongs to an aggregate transaction for which another subtransaction was already aborted.

We define $Ns(p)$ - the state of the node p as a sequence of subtransactions $s_1 \dots s_m$ corresponding to the product defined by p. For a node p, $Ns(p)$ is calculated depending on the $p \rightarrow info$ as follows:

- if $p \rightarrow info = Pid_i$, where $1 \leq i \leq n$, then $Ns(p)$ is returned by the the $SAgree(C', M_i, Pid_i)$ sub-protocol that will be detailed below. So, if the subtransaction in that C' buys from M_i the product with the identifier Pid_i successfully finished *Agreement*, then $Ns(p) = (Sid_i, C', M_i, Pid_i, 1)$, otherwise $Ns(p) = (Sid_i, C', M_i, Pid_i, abort)$.

  The *SAgree* sub-protocol will be detailed below in Table 3.

- if $p \rightarrow info = \vee$, then

$$Ns(p) = \begin{cases} Ns(l), & \text{if } \exists\, l, \text{ the leftmost child of } p \text{ such that } s.fst = 1, \text{ for all } s \in Ns(l) \\ Ns(r), & \text{otherwise} \end{cases}$$

  where r is the rightmost child of p.

  The node p corresponds to $\vee$ operator w.r.t. the customer's choices and these preferences are prioritized by appearance in the child nodes of p from left to the right. $Ns(p)$ is the node state of the the leftmost child of p, denoted $Ns(l)$, for which all subtransactions from $Ns(l)$ have successfully passed *Agreement*. Otherwise, if all subtransactions from node states of all children of p are aborted, then $Ns(p)$ is the node state of the rightmost child of p.

- if $p \rightarrow info = \wedge$, and $c_1, \dots, c_k$ are all children of p, then we have two cases:

  1. if $s.fst = 1$, for any s from $Ns(c_j)$, for any $1 \leq j \leq k$, then $Ns(p) = Ns(c_1) \dots Ns(c_k)$.
  2. otherwise, let c_j be, where $1 \leq j \leq k$, the leftmost child of p with $Ns(c_j) = s_{j1} \dots s_{jm}$ such that $s_{jl}.fst = abort$, for all $1 \leq l \leq m$. In this case, $Ns(p) = Ns(c_1) \dots Ns(c_j)$. Even if the subtransactions from $Ns(c_1)$,..., $Ns(c_{j-1})$ have successfully passed *Agreement*, the aborted subtransactions from $Ns(c_j)$ lead to aborting the entire aggregate transaction corresponding to p. That is why all subtransactions from $Ns(c_1), \dots, Ns(c_{j-1})$ will be aborted. Thus, will set $s.fst = abort$, for any $s \in Ns(c_r)$, for any $1 \leq r \leq j-1$.

  Because the node p corresponds to $\wedge$ operator, $Ns(p)$ is the sequence of node states of p's children. The sequence of node states of p's children is efficiently calculated until c_j the leftmost child of p for that $Ns(c_j)$ contains only aborted subtransactions.

Thus, $Ns(p)$ is a sequence of subtransactions in which either all subtransactions successfully passed *Agreement* or all subtransactions are aborted. Examples of node states computation are presented in Appendix 5.

The *Agreement* sub-protocol, described in Table 2, recursively calculates $Ns(t)$ (t is the root of the tree derived from the customer's choices), traversing the tree in a similar manner with depth-first search. For any node p of the tree, we use a *child* array to store the node states of all children of p.

Initially, before the application the *Agreement* sub-protocol in the first round of the protocol, $Ns(p) = \lambda$, for any node p of the tree t (λ is empty string). If the protocol does not terminate after the first round, then before starting a future round, the states of the nodes of tree are: some nodes q that corresponds to the product identifiers have in $Ns(q)$ a unique aborted subtransaction (because some substransactions have been aborted in previous rounds) and all other nodes p have $Ns(p) = \lambda$. For efficiency, in future rounds, the protocol will not apply *Agreement* to those nodes where their state is an aborted subtransaction (as we can see in the lines 1, 6).

At the lines 2-3, if some conditions are met, the protocol computes $Ns(p)$ for a node p, depending on

Table 2: Agreement sub-protocol.

```
Agreement(t)
1. if (t→left ≠ NULL and s.fst ≠ abort, for s ∈ Ns(t→left)) child[0] = Agreement(t→left);
2. if ((t→info = ∨ and s.fst = 1, for all s from child[0]) or
3.     (t→info = ∧ and s.fst = abort, for all s from child[0])) Ns(t) = child[0];  return Ns(t);
4. j = 1;  k = t→left→right;
5. while (k ≠ NULL)
6.     if (s.fst ≠ abort, for s from Ns(k))  child[j] = Agreement(k);
7.     if (t→info = ∨ and s.fst = 1, for all s from child[j])  Ns(t) = child[j];  return Ns(t);
8.     if (t→info = ∧ and s.fst = abort, for all s from child[j])
9.         for (c = 0; c ≤ j; c = c + 1)  Ns(t) = Ns(t)child[c];  end for
10.        for (all s ∈ Ns(t) such that s.fst ≠ abort)  AggAbort(s);  s.fst = abort; end for
11.        return Ns(t);
12.    k = k→right;  j = j + 1;   end while
13. if (t→info = Pid_i)  Ns(t) = SAgree(C', M_i, Pid_i);  return Ns(t);
14. else if (t→info = ∨)  k = t →left;
15.                       while (k→right ≠ NULL)  k = k→right;  end while
16.                       Ns(t) = Ns(k);  return Ns(t);
17.        else  for (c = 0; c ≤ j - 1; c = c + 1)  Ns(t) = Ns(t)child[c];  end for
18.              return Ns(t);   end if
19. end if
```

the node state of the left most child of p. For a node p with a least two children, the while loop (lines 5-12) computes the node state of any child of p except the left most one. The way in which node state is computed is essential to obtain the fair exchange and atomicity of a complex transaction (lines 7-11): if an aborted subtransaction/sequence of subtransactions leads to aborting the entire aggregate transaction, but some subtransactions from the aggregate transaction successfully completed *Agreement*, then the ones that are successfully must also be stored in the node state corresponding to ∧ operator (line 9) and afterwards aborted (line 10). In this case, for each subtransaction s that successfully completed *Agreement*, the $AggAbort(s)$ procedure will be initiated by TTP to cancel the coin involved in s in the same manner with the messages $1.6'.i$-$1.7'i$ from *Aborting* procedure (Table 3), and the flag of s is set on abort.

For a node with a product identifier as *info*(line 13), the protocol applies the *SAgree* sub-protocol in that the customer establishes the agreement conditions with the merchant for buying the product. The node state for a node that corresponds to ∨ operator, for which all subtransactions from all its children are aborted, is computed at the lines 14-16. The node state for a node that corresponds to ∧ operator for that all subtransactions from all its children successfully completed *Agreement*, is computed at the lines 17-18.

The subtransactions that are initiated in a protocol round can become aborted in different phases of the protocol without solving the customer's options. But, because the customer can have many options w.r.t. the products he wants to buy, new rounds of the protocol must be executed in which are not considered any more the products that are information in the nodes for that the node states have a substransaction already aborted in a previous round.

Next, we will give details about $SAgree(C',M_i,Pid_i)$ sub-protocol presented in Table 3. In this sub-protocol C buys a digital coin, validates the coin to TTP, and establishes the agreement with M_i on the subtransaction's terms.

C generates a new digital coin that is a number c_i of 256 bits consisting of a unique coin serial number represented on the first 224 bits and the coin value in the last 32 bits. The protocol starts with running the GBDS protocol between C and CB on the coin c_i. CB transfers the coin value from C_{acct} to the commit-buffer, and after running all steps of the GBDS protocol, C obtains $sig_{CB}(c_i)$-the signature of CB on c_i on behalf of the bank's group. CB doesn't know the serial number of c_i because his signature on c_i is blind, and only knows the identity of the customer and the value of some digital coin purchased by him.

After C gets the group signature on c_i, he validates at TTP an encrypted version of c_i. C ge-

Table 3: SAgree sub-protocol.

SAgree(C', M_i, Pid_i)
1.1.*i*. $C' \rightarrow TTP$: Sid_i, $\{Sid_i, C', M_i, C'_{ipub}, K_i\}_{TTP_{pub}}$, $\{c_i\}_{K_i}$, $sig_{CB}(c_i)$, $sig_{C'}(sig_{CB}(c_i))$
1.2.*i*. $TTP \rightarrow C'$: Sid_i, T_{iTTP}, L_i, N_{iTTP}, S_{iTTP}
where $S_{iTTP} = sig_{TTP}(Sid_i, C', M_i, \{c_i\}_{K_i}, V_i, T_{iTTP}, L_i, N_{iTTP})$
1.3.*i*. $C' \rightarrow M_i$: Po_i, $sig_{C'}(Po_i)$, $\{c_i\}_{K_i}$, V_i, T_{iTTP}, L_i, N_{iTTP}, S_{iTTP}
where $Po_i = Sid_i, C', M_i, Pid_i, Pr_i, Q_i, V_i, DC_{addr}, h(N_{iC}), C'_{ipub}$
if (M_i_*agreement*) 1.4.*i*. $M_i \rightarrow C'$: Sid_i, $sig_{M_i}(sig_{C'}(Po_i))$, $\{M'_{ipub}\}_{C'_{ipub}}$
return $(Sid_i, C', M_i, Pid_i, 1)$
else if (1.4′.*i*. $M_i \rightarrow C'$: Sid_i, $sig_{M_i}(Po_i, abort)$) *Aborting*(s_i) **end if** **end if**

where the *Aborting*(s_i) procedure consist of:
1.5′.*i*. $C' \rightarrow TTP$: Po_i, $sig_{C'}(Po_i)$, $\{c_i\}_{K_i}$, V_i, T_{iTTP}, L_i, N_{iTTP}, S_{iTTP}
1.6′.*i*. $TTP \rightarrow C'$: Sid_i, *cancel*, $sig_{TTP}(Sid_i, c_i, sig_{CB}(c_i), cancel)$
1.7′.*i*. $C \rightarrow CB$: Sid_i, *cancel*, $\{c_i, sig_{CB}(c_i), C, C_{acct}\}_{CB_{pub}}$, $sig_{TTP}(Sid_i, c_i, sig_{CB}(c_i), cancel)$
return $(Sid_i, C', M_i, Pid_i, abort)$

nerates a nonce Sid_i as subtransaction identifier, a symmetric key K_i and sends to TTP the message 1.1.*i*. TTP obtains C'_{ipub}, K_i by decrypting $\{Sid_i, C', M_i, C'_{ipub}, K_i\}_{TTP_{pub}}$, obtains c_i by decrypting $\{c_i\}_{K_i}$, and uses C'_{ipub} to verify the signature of the customer on the signed coin. TTP checks the validity of c_i by verifying $sig_{CB}(c_i)$ using the group public key, and checks whether c_i has already been spent, or validated by TTP (in a previous request) but not yet spent or canceled, by verifying the *spent* flag of the c_i's serial in the global list of coin's serial. If all checks out, TTP adds c_i in the list setting the c_i's *spent* flag on 0, and sends to C the message 1.2.*i*. The message 1.2.*i* contains a timestamp T_{iTTP}, a lifetime of encrypted coin's validity L_i, a nonce N_{iTTP} all to avoid replay attacks, and S_{iTTP} - the signature of TTP. On reception, C checks if T_{iTTP} and L_i are recently enough, and then verifies S_{iTTP}.

C initiates the agreement by sending to M_i the message 1.3.*i*. Po_i contains $h(N_{iC})$ whose goal is to be used as a barcode on the product and is set by M_i such that only who knows the password N_{iC} can collect the product from DC_{addr}; N_{iC} is kept secret by C, while M_i receives $h(N_{iC})$.

On reception of the message 1.3.*i*, M_i verifies Po_i, if T_{iTTP} and L_i are recently enough, the signature of C on Po_i, and S_{iTTP}. If M_i is satisfied (meaning that the value of the boolean variable M_i_*agreement* is 1), S_{iTTP} assures him that $\{c_i\}_{K_i}$ represents the encryption of a valid coin (that was signed by a bank from the group, and its lifetime has not expired) of value V_i from Po_i. Thus, M_i sends to C the message 1.4.*i* to ensure C by M_i's agreement. After reaching the agreement, $s_i = (Sid_i, C', M_i, Pid_i, 1)$ is returned as the state of the node with *info* Pid_i. If M_i is not satisfied, he sends an abort message 1.4′.*i* to C who applies *Aborting*(s_i) procedure. In *Aborting*(s_i), C cancels c_i in the messages 1.5′.*i*-1.7′.*i* and $s_i = (Sid_i, C', M_i, Pid_i, abort)$ is returned as the state of the node with *info* Pid_i. To cancel the coin, C sends 1.5′.*i* to TTP that checks it and sends back a cancellation request in 1.6′.*i*. Further, C sends this request to CB who checks $sig_{CB}(c_i)$, TTP's signature and if c_i has not already been spent or canceled. If all checks are satisfied, CB sets the *spent* flag of c_i to 2, transfers the coin value from commit-buffer to C_{acct}, and sends to C a signed acknowledgment of successfully cancellation of c_i. So, c_i's value is redeemed by C. If after M_i receives 1.3.*i*, he doesn't continue the sub-protocol, C will apply *Aborting*(s_i).

After *Agreement* is completed, $Ns(t - root)$ is stored in the variable Ps that indicates the protocol state. So, Ps is the sequence of the subtransactions for which all successfully completed *Agreement* or all aborted.

3.5 Delivery Sub-protocol

If all subtransactions from Ps successfully completed *Agreement*, then *Delivery* can be started to physically deliver the products involved in any subtransaction from Ps

In the Table 4 we give details about *Delivery* subprotocol for an arbitrary subtransaction s_i. The product P_i has a barcode $h(N_{iC})$ set by M_i to control the access of C to DC. P_i with this barcode is taken by

Table 4: Delivery sub-protocol.

2.1.*i*. $M_i \rightarrow DA_i$: Sid_i, Pid_i, DC_{addr}, $\{M_i, M'_{ipub}\}_{DA_{ipub}}$, $sig_{M_i}(Sid_i, M_i, Pid_i, DC_{addr})$
2.2.*i*. $M_i \rightarrow DA_i$: Sid_i, P_i
2.3.*i*. $DA_i \rightarrow M_i$: Sid_i, $sig_{DA_i}(Sid_i, M_i, Pid_i, DA_i, DC_{addr})$
2.4.*i*. $DA_i \rightarrow DC$: Sid_i, P_i
2.5.*i*. $DC \rightarrow DA_i \rightarrow M_i \rightarrow C'$: Sid_i, $sig_{DC}(Sid_i, M_i, Pid_i, DA_i, DC_{addr})$
2.6.*i*. $DC \rightarrow C'$: Sid_i, P_i
2.7.*i*. $C' \rightarrow DC$: Sid_i, $sig_{C'}(Sid_i, M_i, Pid_i, DC_{addr}, C', N_{iC})$

DA_i from M_i to be delivered. M_i sends to DA_i a delivery request message 2.1.*i*. DA_i obtains M'_{ipub} and checks the signature of M_i. In the message 2.2.*i*, DA_i collects the product P_i from M_i. To confirm the collection of P_i, DA_i sends to M_i an acknowledgment in the message 2.3.*i*. In the message 2.4.*i*, DA_i posts P_i to DC. Upon receiving P_i from DA_i, DC confirms him in the message 2.5.*i* by a signed acknowledgment which DA_i forwards to M_i, and M_i to C. C collects P_i from DC using N_{iC} and checks if the collected product meets the specifications from Po_i. If C is satisfied, he sends in the message 2.7.*i* a signed acknowledgment to DC.

If all subtransactions from Ps successfully completed *Delivery*, then each subtransaction's flag from Ps is set to 2.

3.6 Payment Sub-protocol

If C collects all products involved in any subtransaction from Ps and is satisfied, then he sends the payment for each product to the suitable merchant. Below, we present the *Payment* sub-protocol for an arbitrary subtransaction s_i from Ps.

3.1.*i*. $C' \rightarrow M_i$: $Sid_i, \{K_i\}_{M'_{ipub}}, sig_{C'}(K_i), sig_{CB}(c_i)$

M_i obtains K_i, verifies $sig_{C'}(K_i)$, and recovers c_i by decrypting with K_i the encrypted coin received in the message 1.3.*i*. M_i verifies the validity of $sig_{CB}(c_i)$ and, if c_i is valid, he sends it to M_iB in the message 3.2.*i* for redemption.

3.2.*i* : $M_i \rightarrow M_iB$: $Sid_i, \{co_i, sig_{CB}(co_i),$
$sig_{M_i}(co_i, sig_{CB}(co_i)), M_i, M_{iacct}\}_{M_iB_{pub}}$

M_iB checks M_i's signature, checks $sig_{CB}(c_i)$ and uses the global list of the coins to check if c_i has already been spent or canceled. If all checks are satisfied, M_iB updates the global list by setting the *spent* flag of c_i on 1, transfers the c_i's value from commit-buffer to M_{iacct}, and sends to M_i a signed acknowledgment of successfully redemption of c_i; otherwise, M_iB sends to M_i an error message.

3.3.*i*. $M_iB \rightarrow M_i$: $sig_{M_iB}(ack)$

If all subtransactions from Ps successfully completed *Payment*, then each subtransaction's flag from Ps is set to 3.

If in a subtransaction s_i from Ps, C does not send to M_i the decryption key K_i of the encrypted coin or sends to M_i in 3.1.*i* a wrong decryption key, then M_i sends to TTP all the messages received/sent from/to C in s_i. TTP checks the messages of s_i and if all checks are successfully, then sends K_i to M_i, the key that is in possession of TTP from the *Agreement* sub-protocol: $TTP \rightarrow M_i$: $Sid_i, \{K_i\}_{M'_{ipub}}, sig_{TTP}(K_i), sig_{CB}(c_i)$. M_i decrypts c_i using K_i, checks the validity of $sig_{CB}(c_i)$ and continues the sub-protocol with 3.2.*i*.

3.7 The Complex Transaction Protocol

Next, we describe the protocol for complex transactions, presented in Table 5, that relies on the three phases discussed above. t is the root of the tree derived from the choices of C. Initially, $Empty(Ns(t))$ sets $Ns(p) = \lambda$, for any node p of the tree t. A **while** iteration executes a round of the protocol, that executes the three phases of the protocol. At the line 3, Ps variable retains the sequence of subtransactions that have run *Agreement* corresponding to options provided by C. If all subtransactions from Ps successfully completed *Agreement*, then *Delivery* may take place to physically deliver the products involved in any subtransaction from Ps. At the line 5, Ps retains the sequence of subtransactions that have run *Delivery*. Further, if all subtransactions from Ps successfully completed *Delivery*, then *Payment* takes place and the sequence of subtransactions that solves the options of C is returned.

If after *Agreement* took place, Ps is a sequence of subtransactions that are all aborted, then the options of C can not be successfully solved. In this case, at the line 10, the subtransactions sequence will be returned

Table 5: The Complex Transaction Protocol.

```
1.  Empty(Ns(t));
2.  while (1)
3.        Ps = Agreement(t);
4.        if (s.fst = 1, for all s from Ps)
5.              Ps = Delivery(Ps);
6.              if (s.fst = 2, for all s from Ps)
7.                    Ps = Payment(Ps); return Ps;
8.              else  Aborting(s), for all s from Ps;
9.                    Update(t);  end if
10.       else  return Ps;  end if  end while
```

as an evidence of the protocol's failure.

The case in that after *Agreement* took place, all subtransactions $s_1 \dots s_k$ from *Ps* successfully completed *Agreement*, but some of these subtransactions did not successfully completed *Delivery* of physical products (e.g. some merchant doesn't post the product or posts a wrong product), is treated at the line 8. In this case, the $Aborting(s_i)$ procedure is applied for all $1 \leq i \leq k$, to abort all the substransactions from *Ps* that successfully completed *Agreement*. According to assumption 7 from the section 3.1, *DC* has a device that allows *C* to send to TTP the encrypted recording of the moment when *C* receives products that are not in conformity with the agreement established. The identifiers of these products are stored in the buffer *PidsAborted* used at the line 9 to update the tree with the root t. $Update(t)$ updates $Ns(p)$, for any node p of the tree, as follows:

- if $p \rightarrow info = Pid_i$, with $1 \leq i \leq n$, then we have the following cases: if $Ns(p)$ has the flag on *abort*, then $Ns(p)$ remains unchanged; if $p \rightarrow info \in PidsAborted$, then $Ns(p) = (Sid_i, C', M_i, Pid_i, abort)$; in all other cases $Ns(p) = \lambda$.
- if $p \rightarrow info \neq Pid_i$, with $1 \leq i \leq n$, then $Ns(p) = \lambda$.

By $Update(t)$, the states of the nodes corresponding to products from aborted subtransactions in the current protocol's round are mentained in the tree with the root t, so that these products are not taken into consideration in a new protocol's round.

In a new round (**while** iteration), the protocol searches a new sequence of subtransactions that will successfully finish all three sub-protocols, and in this way satisfying the options of *C*. The protocol terminates when encounters a round in which the protocol state *Ps* computed after that round contains a sequence of subtransactions that solves *C*'s options, or when encounters a round in that *Ps* computed after *Agreement* has all subtransactions aborted. In the last case, the protocol can not solve the options of *C*.

4 SECURITY ANALYSIS

4.1 Fair-exchange

For a tree t that describes the choices of *C*, we define $solve(t)$ - the sequence of products obtained by *C* after the protocol execution and that solves the choices of *C*. $solve(t)$ is defined depending on the protocol state *Ps* calculated after the protocol execution.

If after a protocol's round $Ps = s_1 \dots s_m$ such that $s_i.fst = 3$ for any $1 \leq i \leq m$, then $solve(t) = Pid_1 \dots Pid_m$ meaning the products from all subtransactions that successfully terminates all three subprotocols. Otherwise, if in a protocol's round after *Agreement* sub-protocol, *Ps* is a sequence of aborted subtransactions, then $solve(t) = abort$ meaning that the protocol can not solve the options of *C*.

Our protocol assures fair exchange if after the protocol execution, either *C* gets the sequence of physical products $solve(t) = Pid_1 \dots Pid_m$ and each M_i gets the payment for the product Pid_i, where $1 \leq i \leq m$, or none do. From the construction of our protocol, after its execution, *C* can obtain either a sequence of products that solves his choices, or nothing. Furthermore, after the protocol execution, *C* can 't obtain a sequence of products $Pid_1 \dots Pid_k \neq solve(t)$.

If *C* and each M_i, where $1 \leq i \leq n$, behave honestly, the proposed protocol assures fair exchange. In what follows, we will consider all possible scenarios in which any M_i or *C* behave dishonest or prematurely abort the protocol.

If *C* behaves dishonest, then the following scenarios are possible:

1. After *Delivery* sub-protocol took place in some protocol's round, $Ps = s_1 \dots s_k$ is a sequence of subtransactions that successfully completed *Delivery* sub-protocol. So, *C* collected all products involved in any subtransaction from *Ps* and he is satisfied. However, *Payment* sub-protocol, for a subtransaction s_i from *Ps*, where $1 \leq i \leq k$, *C* does not send to M_i the decryption key of the encrypted coin or sends to M_i a wrong decryption key. This scenario is solved as mentioned in *Payment* subprotocol, TTP providing to M_i the correct key.
2. *C* sends the same coin to TTP (in 1.1.*i*) in two different sessions of *Agreement* sub-protocol, to initiate two different subtransactions s_i, s_j with two distinct merchants, in the same round or in different rounds of the protocol. This scenario

is solved because all banks and TTP maintain a global list of coin's serial already spent, validated but unspent, or canceled. On reception from C of the first request for coin validation, TTP adds the coin to the list, and any new validation request of the same coin from C is detected by TTP that aborts s_j by $s_j.fst = abort$.

3. In a subtransaction s_i from *Agreement* sub-protocol, C sends to M_i in $1.3.i$, an encrypted coin already spent, or already canceled or of insufficient value. If the coin is already spent but wasn't used to buy from M_i, then M_i detects this by verifying S_{iTTP}. Otherwise, if the coin was already used to buy from M_i, then M_i can check this by verifying Sid_i and N_{iTTP}. If the coin was already canceled, then M_i detects this because T_{iTTP} and L_i are not recently enough. If the coin is of insufficient value, M_i detects this by checking if the value from Po_i is equal with the encrypted coin's value validated by TTP. So, M_i detects double spending/double cancelling/insufficient coin's value from C, and aborts s_i by $s_i.fst = abort$.
4. In *Agreement* sub-protocol, C sends to CB $1.7'.i$ many times for multiple redemption of the same canceled coin. This scenario is solved because CB checks if the coin received in a cancellation request has already been canceled.

If M_i behaves dishonest, then the following scenarios are possible:

1. In a subtransaction s_i from *Agreement* sub-protocol, M_i receives from C' a correct message $1.3.i$, but he sends to C' an abort message $1.4'.i$ or doesn't continue the sub-protocol. Such behavior brings no benefit to M_i because he is in possession of an encrypted coin with a key that does not know, so he can't get the payment. But C has bought a coin which can not be used by him. In both scenarios, C applies $Aborting(s_i)$ to cancel and to redeem the coin, and to abort s_i. If the aborted s_i is a component of an aggregate transaction, then all other subtransactions from the aggregate that successfully completed *Agreement* sub-protocol will be aborted by *AggAbort* procedure and by setting their flags on *abort*.
2. After *Agreement* sub-protocol took place, $Ps = s_1 \dots s_k$ is a sequence of subtransactions that successfully completed *Agreement* sub-protocol. In s_i from Ps, M_i sent $1.4.i$ to C', but in *Delivery* sub-protocol he doesn't post the product or posts a product that doesn't comply with the specifications from Po_i. M_i does not have any benefit from this behavior, but can prejudice the other honest merchants involved in $s_1 \dots s_k$. In s_i, C pushes the button of the DC's device that allows sending to TTP the recording of the moment when C unwraps the packed product, proving to TTP that the product is wrong. Because the product from s_i is not complying with the specifications from Po_i, the rest of the products involved in the sequence $s_1 \dots s_k$ can't lead to $solve(t)$. So, C sends to TTP all the messages received/sent from/to each M_j, where $1 \leq j \leq k$. TTP checks the information received from C, applies $Aborting(s_j)$, where $1 \leq j \leq k$ and triggers an off-line procedure sending to M_i the proof of his dishonest behavior. Also, TTP requests from M_i the payment for the transportation services of products provided by the honest merchants to DC and back. After TTP receives the payment, he returns the transportation costs to each honest merchant from $s_1 \dots s_k$.
3. In *Payment* sub-protocol, M_i sends many times the same $3.2.i$ for multiple redemption of the same coin. This scenario is solved by M_iB checking if the coin received in $3.2.i$ has already been spent.

Each party involved in the protocol must keep a record of every message sent or received in protocol including signed acknowledgments of DC and DA_i, where $1 \leq i \leq n$. DC and DA_i, where $1 \leq i \leq n$, have no interest not to follow the protocol steps, because their interest is to get profit from fees for such services provided in e-commerce transactions. However, if one of the parties mentioned above behaves dishonest, the other parties send the records to TTP to trigger off-line mechanisms to ensure fairness.

4.2 Anonymity

Our protocol ensures the customer's anonymity if no party and no coalition between parties can make a link between the true identity of the customer, C, and the pseudo identity of the customer, C', which he uses in protocol.

In our protocol, we use an electronic cash payment based on group blind digital signatures on behalf of the banks. The only steps from our protocol in that the customer uses his true identity are the GBDS protocol's steps and in the message $1.7'.i$ in that the customer cancel his coin, because CB must know C_{acct} to charge it with the coin's value or to redeem the coin's value. In the GBDS protocol, CB knows only that C bought a coin with a certain value, but it doesn't know the serial of the coin. Following, CB can't associate C with the coin bought by him, maintaining thus the anonymity of the customer. Also, in the message $1.7'.i$, the customer reveals his true identity only to CB, but this does not destroy the customer's anonymity because the coin will be canceled and not used

Table 6: Informations that each party knows after protocol execution.

	Entity					
Info	C	M_i	CB	M_iB	DA_i	DC
C	y	n	y	n	n	n
M_i	y	y	n	y	y	n
CB	y	n	y	n	n	n
M_iB	n	y	n	y	n	n
DA_i	n	y	n	n	y	y
DC	y	y	n	n	y	y
C'	y	y	n	n	n	y
c_i	y	y	n	y	n	n
$C\&s_i$	y	n	n	n	n	n
$C'\&s_i$	y	y	n	n	n	y

anymore. Another feature of the GBDS protocol is the anonymity of CB: any party can check if $sig_{CB}(c_i)$ is valid, without knowing who is the bank that signed the coin c_i. CB is not known by any other party (except C), so, CB can't participate in no coalition with any other party to destroy the customer's anonymity. To ensure the customer's anonymity when C collects the physical products, our protocol doesn't use the customer's correspondence address but uses a destination cabinet where the product is placed.

We show in Table 6, the information that each party in the protocol knows after protocol execution. The information have the following meaning. For example, we consider the first row: y under the column C and CB means that C and CB know C - the true identity of the customer; n under the column M_i, M_iB, DA_i and DC means that the true identity of the customer is not known to M_i, M_iB, DA_i and DC. $C\&s_i$ means that C performs the subtransaction s_i. The meaning is extended for $C'\&s_i$.

From the Table 6, we observe that no party alone has sufficient information to link the true identity of the customer, C, with the pseudo identity C'. Only C can disclose this information if he wants.

We analyze the coalition of the maximum size that can be formed by parties involved in protocol to try to destroy the customer's anonymity. The coalition of the maximum size consists of the following parts: $M_1,\ldots,M_n,M_1B,\ldots,M_nB,DA_1,\ldots,DA_n$, and DC. For any $1 \leq i \leq n$, DA_i has information only about entities M_i, and DC, and no information about C. For any $1 \leq i \leq n$, M_iB have no information about C. DC has the information $C'\&s_1,\ldots,C'\&s_n$, but no information about C. For any $1 \leq i \leq n$, M_i has the information $C'\&s_i$. So, the only information related to customer obtained by the coalition of the maximum size is $C'\&s_1,\ldots,C'\&s_n$, but no information about the customer's true identity. Thus, the customer's anonymity is preserved.

5 CONCLUSIONS

Fair exchange in complex transactions is much harder to obtain compare to fair exchange in a subtransaction, because fair exchange obtained in all subtransactions from a complex transaction does not directly implies fair exchange in entire complex transaction. The complexity of our protocol is higher for complex transactions containing both aggregate and optional transactions then for complex transactions containing only aggregate transactions. The requirement for optional transactions impose the protocol execution in multiple rounds to solve the customer's options. We remark that obtaining the customer's anonymity in complex transactions is not simple. To obtain the anonymity of the customer, the challenge in our protocol was to guarantee this requirement both in payment and collection of physical products.

We proposed an e-commerce protocol taking into consideration several requirements that are not considered so far together in literature: physical products delivery, aggregate transaction, optional transaction, fair exchange and anonymity. If some of the requirements mentioned above are not needed in an e-commerce application, then we will use simplified versions of the proposed protocol. Thus, if optional transactions are not required in a complex transaction e-commerce protocol, then only one round of the protocol is necessary and *Agreement* sub-protocol is replaced with customer initiating simultaneously all subtransactions for agreement on terms. If the customer's anonymity is not required, then a simpler payment method that replaces GBDS protocol is used. Also, in this case, the destination cabinet and all messages involving him are eliminated.

Future work will include formal verification of the proposed protocol and extending the protocol by taken into consideration of active intermediary agents (that are not delivery agents) between customer and merchants.

REFERENCES

Alaraj, A. (2012). Fairness in physical products delivery protocol. *International Journal of Computer Networks & Communications (IJCNC)*.

Birjoveanu, C. V. (2015). Anonymity and fair-exchange in e-commerce protocol for physical products deli-

very. In *12th International Conference on Security and Cryptography*. SCITEPRESS.

Djuric, Z. and Gasevic, D. (2015). Feips: A secure fair-exchange payment system for internet transactions. *The Computer Journal*.

Draper-Gil, G., Ferrer-Gomila, J. L., Hinarejos, M. F., and Zhou, J. (2013). An asynchronous optimistic protocol for atomic multi-two-party contract signing. *The Computer Journal,*.

Li, H., Kou, W., and Du, X. (2006). Fair e-commerce protocols without a third party. In *11th IEEE Symposium on Computers and Communications*. IEEE.

Liu, Y. (2009). An optimistic fair protocol for aggregate exchange. In *2nd International Conference on Future Information Technology and Management Engineering*. IEEE.

Lysyanskaya, A. and Ramzan, Z. (1998). Group blind signature: a scalable solution to electronic cash. In *Financial Cryptography*. Springer.

Onieva, J. A., Lopez, J., and Zhou, J. (2009). *Secure Multi-Party Non-Repudiation Protocols and Applications*. Springer.

Zhang, Q., Markantonakis, K., and Mayes, K. (2006). A practical fair exchange e-payment protocol for anonymous purchase and physical delivery. In *4th ACS/IEEE International Conference on Computer Systems and Applications*. IEEE.

APPENDIX

For some nodes of the tree from Figure 1, we will give computation example of node states:

- If p is the node with $p \rightarrow info = Pid_1$, then $Ns(p) = (Sid_1, C', M_1, Pid_1, 1)$ if the subtransaction with identifier Sid_1 successfully finished *Agreement*; otherwise $Ns(p) = (Sid_1, C', M_1, Pid_1, abort)$.
- If q is the first node with $q \rightarrow info = \vee$ then: $Ns(q) = s_1$, where $s_1 = (Sid_1, C', M_1, Pid_1, 1)$, if the subtransaction s_1 successfully finished *Agreement*; otherwise, $Ns(q) = s_2$, where $s_2 = (Sid_2, C', M_2, Pid_2, fst)$ and $s_2.fst \in \{1, abort\}$.
- If t the root node, c_1 the first node (from left) with $c_1 \rightarrow info = \vee$, c_2 the second node with $c_2 \rightarrow info = \vee$, c_3 the node with $c_3 \rightarrow info = Pid_5$ and c_4 the node with $c_4 \rightarrow info = Pid_6$, then:
  - if $Ns(c_1) = s_1$, $Ns(c_2) = s_4$, $Ns(c_3) = s_5$ and $Ns(c_4) = s_6$, where $s_i = (Sid_i, C', M_i, Pid_i, 1)$ with $i \in \{1, 4, 5, 6\}$, then $Ns(t) = s_1 s_4 s_5 s_6$.
  - if $Ns(c_1) = s_1$, $Ns(c_2) = s_4$ as above, but $Ns(c_3) = s_5$ with $s_5 = (Sid_5, C', M_5, Pid_5, abort)$, then $Ns(t) = s_1 s_4 s_5$ and s_1, s_4 will be aborted.