Model Checking a Client-Side Micro Payment Protocol

Kaylash Chaudhary
School of Computing, Information Mathematical Sciences
University of the South Pacific, Fiji
Email: kaylash.chaudhary@usp.ac.fj

Ansgar Fehnker
Formal Methods and Tools
University of Twente, Enschede, Netherlands
Email: ansgar.fehnker@utwente.nl

***Abstract*—Virtual payment systems overcome the drawbacks such as processing and operational cost of the traditional payment system. The main aim of the virtual payment system is to provide efficient services in terms of cost. Online payment using credit card is one of the most expensive of all payment means. This gives advantage to micropayment systems where only small amounts are used for e-commerce. Payment which are small will be costly if paid through credit card. Therefore, there are several micropayment systems that exist and some have been proposed. One of the proposed micropayment system that this paper will talk about is Netpay. We will do model checking to check the correctness of this payment system and to see whether the protocol designers property claim is valid. Correctness is important in payment systems because money is involved in it, therefore the protocol needs to be validated.**

This paper examines the client-side version of the Netpay protocol and provides its formalization as a CSP model. The PAT model checker is used to prove three properties essential for correctness: impossibility of double spending, validity of an ecoin during the execution and the absence of deadlock. We prove that the protocol is executing according to its description based on the assumption that the customers and vendors are cooperative. This is a very strong assumption for system built to prevent abuse, but further analysis suggests that without it the protocol does no longer guarantee all correctness properties. We compare the two variation of the protocol with each other and with the properties claimed by the protocol designers.

I. Introduction

With the growth of internet, most goods are sold online. Virtual payment systems evolved to cater for online transactions. Credit card is one method for performing online transaction but it can be expensive for payments with small amounts. Micropayment technologies emerged to facilitate payments with smaller amounts.

There are many micro-payment systems available for users to buy goods online such as Netpay [8], Millicent [9], Micromint [14], Payword [14], MiniPay [11], Micro-iKP [10] and POPCORN [13]. There are also many micro-payment systems proposed for content sharing in peer to peer networks [3] [4], [16], [17] and [18].

Since micropayment systems deals with online payment, the protocol should be secure. Security is one of the essential property of online payment systems. A payment system should not allow a customer to double spend. Modeling and verification has been performed to verify this double spending property on other protocols such as Bitcoin [1][2]. We have proved that there is no double spending in server-side Netpay protocol [5]. Client-side Netpay is a variation of the Netpay protocol. The difference between the two is e-wallet location. In the client-side, the e-wallet is located at the customer side. Due to e-wallet location, the transactions are different. In server-side, the vendor has to ask for e-wallet location from broker and then request the e-wallet from the vendor. In client-side, the e-wallet stays at the customer but there is touchstone that is used to verify e-coins which floats from broker to vendor and then from vendor to vendor. Hence, the difference between the two modeling exercise.

This paper models the client-side Netpay using CSP. The model in this paper covers the handling of e-coins. The parts of the protocol that deals with ecoin redemption and digital signature is excluded from the model. We assume that the correctness of the cryptographic hash function is guaranteed independently. We will prove that the protocol can guarantee the touchstone location as recorded by the customer is correct at the end of the transaction , that the customer can not spend more than the e-coin is worth, and finally, that the protocol is non-blocking.

Section II introduces the server-side Netpay protocol. CSP language is described in section III. Section IV shows the description of the Netpay protocol using the CSP language. Correctness of this protocol is discussed in Section V. This paper concludes with discussion for future research in section VI.

II. The Netpay Protocol

The Netpay protocol with server-side e-wallet was proposed by Dai et.al. [6]. It has three types of e-wallets: client side, server-side and cookie-based e-wallet [7]. There are three parties involved in this protocol; customer, vendor and broker. It is assumed that the broker and vendors are honest and are trusted by the customers who may not be honest. To use the protocol, the customers and vendors need to register by opening an account and depositing funds with the broker. The broker is responsible for registration, e-coin generation, debiting and crediting accounts for customers and vendors respectively. The payment is between customers and vendors. Previous work has modeled and verified some properties of server-side Netpay [5]. The difference between the server-side and the client-side Netpay is the location of the e-wallet.

978-1-5090-5753-5/16 $31.00 © 2016 IEEE
DOI 10.1109/APWC.on.CSE.2016.24

Netpay uses a number of cryptography and micro-payment terminologies such as:

- **One-way Hash Function**: Netpay uses this function to generate and verify e-coins. In [7] MD5 was used, but it could be replaced by more secure SHA-1.
- **E-coin**: The one-way hash function is applied repeatedly to a seed to generate a series of paywords called e-coin. The paywords are represented in reverse order with the seed at the end. The length of the e-coin determines its value.
- **E-wallet**: An e-wallet is a database to store e-coins.
- **Seed**: It is a randomly selected value used for e-coin generation.
- **Touchstone**: This is the first payword of the e-coin. It is used to verify e-coins.

Using the one way hash function h, an e-coin $W_1, ..., W_n$ is constructed by applying the hash function $n+1$ times to a seed i.e. $W_0 = h(W_1), W_1 = h(W_2), ..., W_n = h(W_{n+1})$ where W_{n+1} is the seed, and W_0 the touchstone.

The remainder of this section will describe the four basic types of transactions in this protocol. It is assumed that each customer and vendor have a unique ID.

Customer-Broker Transaction The customer sends an e-coin request with parameter n to the broker, who generates e-coins of length n. Each chain has a unique e-coin ID. The broker stores this information in its database, and sends the e-coin ID to the customer.

Customer-Vendor Transaction
If a customer wishes to buy something from the vendor, the customer sends an e-coin ID. The vendor checks if it has the e-coin and verifies it. If the verification is successful, the customer is notified. If the vendor does not have the e-coin, it requests the location from the broker. The broker will reply with the location of the e-coin, the vendor requests this e-coin from that vendor or broker. Initially, the broker will have the e-coin and after that it will be transferred from one vendor to another.

Vendor-Vendor Transaction This transaction occurs when one vendor requests an e-coin from another vendor.

Vendor-Broker Transaction
Vendors need to redeem the e-coins spent by customers. The vendor sends the e-coin IDs, touchstones, customer IDs, vendor ID, e-coins and amount to the broker. The broker will verify e-coins and credit the corresponding amount to the vendors account if the spent e-coins are valid. This paper focuses on the spending of e-coins, and omits redemption of e-coins from the model.

a) Properties of the Netpay Protocol: This paper considers three important properties. The first is on the validity of e-coins. Since there will be transfer of an e-coin from one vendor to another, an e-coin should remain valid in this chain of transfer. The second is on preventing double spending. This protocol prohibits a customer to double spend an e-coin at a different or same vendor. The last property is to show absence of deadlocks.

III. Communicating Sequential Processes (CSP)

This paper will use CSP to model the server-side Netpay protocol. CSP is a formal language which is used for describing interaction between different systems using process algebra [12]. Systems are modelled in CSP as processes and events. A process represents a component of a system whereas an event represents the communication between different components or processes. We have used the following syntax for defining processes in this paper. In order to illustrate the syntax, suppose we have two processes (P and Q), two events (a and b) and a channel c.

- `a -> P` : a is an event which is performed by a process and then behaves as process P.
- `P|||Q` : process P and Q are interleaving processes and these processes perform actions independently.
- `c!x -> P` : a process sends a message with parameter x to another process by synchronizing on channel c and then behaves like process P.
- `c?y -> P` : a process receives a message with parameter y sent by another process on synchronized channel c and then behaves like process P.
- `a -> P[]b -> Q` : a process either performs event a and behaves as P or performs event b and behaves as Q.

Channels are used in CSP to pass message to different processes. In our model, we have used synchronous channels. A sender cannot transmit a message unless the receiver is ready to accept it. There are different tools available which uses CSP as language for model checking. One such tool is Protocol Analysis Toolkit (PAT).

PAT is a tool which supports composing, simulating and verifying different systems [15]. It supports different modeling languages and one such language that we have used to model Netpay protocol is CSP. PAT allows to verify various properties for a protocol.

IV. Description of Netpay Protocol using CSP

This section provides models and description for the Server-Side Netpay protocol. The three parties of the protocol: customer, vendor and broker have been modeled as one process each. For simplicity we assume that there is only one broker, while there can be many customers and vendors.

A. Customer Process

Table 1 shows the customer process for the customer CID. Each customer has its own e-wallet. The e-wallet, EWALLET, stores the ecoin ID, the amount and the touchstone location. Recall, that an e-coin is constructed by applying hash function to a seed. Each payword in Netpay is accompanied by an index to record the number of unspent paywords; the amount. The amount will be abstracted in this paper as either ISPOSITIVE or ISZERO. This is because the properties shown in this paper

are independent of hash function and the exact amount. There are two major tasks that a customer performs; buying e-coins from the broker and spending e-coins at a vendor.

The first task for a customer is to buy ecoins from broker on the $BuyCoin$ channel with parameter customer ID, CID. Since the ewallet will be empty initially, this will be the only enabled event. This further meets the precondition, CUS_STATUS[CID] == IDLE, of this event that is the customer should be IDLE to send a message on this channel. After sending this message, the customer will change to the BUYCOIN state. The customer will be in this state until the broker replies on the $SellCoin$ channel. When the broker replies, the customer adds the ecoin ID, amount and the broker ID to EWALLET. The state of the customer changes to IDLE.

The second task models spending ecoins at a vendor (Table 1, lines 24 to 31). If customer has e-coins and is in the IDLE state, it can buy goods from a vendor on channel $Spend$, which will synchronize with one of the vendors. The channel has four parameters: customer ID, e-coin ID, amount and touchstone location. After this event, the customer will change the state to SPENDING. The vendor will reply either on channel, $Approval$ or $Disapproval$. If there are unspent coins, the customer will update its e-wallet by setting the touchstone location to the vendor ID who replied on channel, $Approval$ (Table 1, lines 33 to 47). If the payment was not accepted, the customer will receive reply on $Disapproval$. In either case, the state of the customer will change to IDLE.

Note, that the customer is the non-trusted party in this protocol. The customer can spend an ecoin as often as it wants to. The vendor and broker should be able to prevent a customer from double spending.

B. Broker Process

The broker process, $Broker$(BID), shown in Table 2 models the broker with unique broker ID. The broker has a database, BROKERDB, which stores the ecoin ID and amount. The broker can be any of the three states; IDLE, BUYCOIN or REQSTONE. The state of the broker is tracked using variable BRO_STATUS. Variables *Beid*, *Bvid* and *Bcid* are used to store intermediate results while generating e-coins or replying to request from customers or vendors.

The broker performs two tasks. The first is generation of ecoins. The broker will receive request from a customer to generate ecoins on channel $BuyCoin$. This can only happen if the broker is in IDLE state, i.e. if BRO_STATUS == IDLE. The state of the broker changes to BUYCOIN. The broker replies on channel $SellCoin$ with parameters ecoin ID and amount. This event changes the state of the broker process to IDLE.

The second task is to provide the touchstone to vendor. This will only happen when the broker generates ecoins. The broker ID will be used as the touchstone location for all fresh ecoins that has not been partially spent. A vendor requests touchstone location on channel $ReqTouchStone$ with two parameters; vendor ID and ecoin ID. The broker searches the database, BROKERDB, and stores the result in *Beid*. The state of the broker changes to REQSTONE. The broker replies with the result on $SendTouchStone$ channel changing the state to IDLE.

Table 1 Customer Process

```
Customer(CID) = [CUS_STATUS[CID] == IDLE &&
    (||y:{0..(MAXCOINS-1)}@(EWALLET[CID][y][0]==-1))]
    BuyCoin!CID
        {
            CUS_STATUS[CID] = BUYCOIN
        }->Customer(CID)
    [][CUS_STATUS[CID] == BUYCOIN]
    SellCoin[CID]?eid1.bid
        {
            var index = 0;
            while(index < MAXCOINS)
            {
                if (EWALLET[CID][index][0] == -1)
                {
                  EWALLET[CID][index][0] = eid1 ;
                  EWALLET[CID][index][1] = ISPOSITIVE;
                  EWALLET[CID][index][2] = bid;
                  index = MAXCOINS;
                }
                index = index + 1;
            }
            CUS_STATUS[CID]=IDLE
        }->Customer(CID)
    []([]x:{0..VENDORS-1};z:{0..MAXCOINS-1}
    @([CUS_STATUS[CID] == IDLE &&
    EWALLET[CID][z][0]!=-1]
    Spend[x]!CID.EWALLET[CID][z][0].
    EWALLET[CID][z][1].EWALLET[CID][z][2]
        {
            CUS_STATUS[CID]=SPENDING
        }->Customer(CID)))
    [][CUS_STATUS[CID] == SPENDING]
    Approval[CID]?vid.eid1.amt
        {
            var index = 0;
            while(index < MAXCOINS)
            {
                if(EWALLET[CID][index][0]==eid1)
                {
                    EWALLET[CID][index][1] = amt;
                    EWALLET[CID][index][2] = vid;
                    index = MAXCOINS;
                }
                index = index + 1;
            }
            CUS_STATUS[CID]=IDLE
        }->Customer(CID)
    [][CUS_STATUS[CID] == SPENDING]
    Disapproval[CID]?vid.eid1
        {
            var index = 0;
            while(index < MAXCOINS)
            {
                if(EWALLET[CID][index][0]==eid1)
                {
                    EWALLET[CID][index][2] = vid;
                    index = MAXCOINS;
                }
                index = index + 1;
            }
            CUS_STATUS[CID] = IDLE;
        }->Customer(CID);
```

C. Vendor Process

The process $Vendor$(VID) shown in Tables 3 and 4 models a vendor with ID VID. The vendor has a database, VENDORDB, which is used to store touchstone. Variable VEN_STATUS is

Table 2 Broker Process

```
Broker(BID) =          [BRO_STATUS == IDLE &&
    (||x:{0..(BROKERDBCONST-1)}
    @(BROKERDB[x][0]==-1))]
    BuyCoin?cid
        {
            Bcid = cid;
            var index = 0;
            while(index < BROKERDBCONST)
            {
                if (BROKERDB[index][0] == eid)
                {
                    eid = (eid+1)%MAXEID;
                    index = BROKERDBCONST;
                }
                index = index + 1;
            }
            BRO_STATUS = BUYCOIN
        }->Broker(BID)
    [][BRO_STATUS == BUYCOIN]
    SellCoin[Bcid]!eid.BID
        {
            Bcid = 0;
            var index = 0;
            while(index < BROKERDBCONST)
            {
                if (BROKERDB[index][0] == -1 &&
                     BROKERDB[index][1] == -1)
                {
                  BROKERDB[index][0] = eid;
                  BROKERDB[index][1] = ISPOSITIVE;
                  index = BROKERDBCONST;
                }
                index = index + 1;
            }
            eid=(eid+1)%MAXEID;
            BRO_STATUS = IDLE
        }->Broker(BID)
    [][BRO_STATUS == IDLE]
    ReqTouchStone[BID]?vid.eid1
        {
            Bvid = vid;
            var index = 0;
            var flag = false;
            while(index < BROKERDBCONST)
            {
                if (BROKERDB[index][0] == eid1)
                {
                    flag = true;
                    index = BROKERDBCONST;
                }
                index = index + 1;
            }
            if(flag == false)
                Beid = -1;
            else
                Beid = eid1;

            BRO_STATUS = REQTSTONE
        }->Broker(BID)
    [][BRO_STATUS == REQTSTONE]
    SendTouchStone[Bvid]!BID.Beid
        {
            Bvid = -1;
            Beid = -1;
            BRO_STATUS = IDLE
        }->Broker(BID);
```

used to track the status of the vendor. A vendor can be in IDLE, SPENDING, REQSTONE, VERIFICATION, VERIFIED or RECVREQ state. Variables VEID, VAMT, VCID, VTSLOC and VVID are used to store intermediate results during communication with broker or customer.

The vendor performs two major tasks; verifying ecoins received from customer and providing touchstone to requesting vendor.

The first task is initiated by the customer, cid. The vendor receives e-coin ID, eid, the amount, amt and touchstone location, $tsloc$, from customer on channel $Spend$. The vendor changes to REQSTONE status if the vendor does not have the touchstone ($tsloc! = VID$) otherwise the vendor changes to status VERIFIED . Recall, a touchstone is used to verify ecoins in the Netpay protocol. If the vendor is in REQSTONE status, the vendor requests touchstone from vendor or broker $Vtsloc[VID]$ on channel $ReqTouchStone$. Initially, if the ecoins in the chain has not been spent, the broker will have the touchstone location. Therefore, the vendoe will request touchstone location from the broker. The state vendor changes to VERIFICATION. The vendor $Vtsloc[VID]$ replies on channel $SendTouchStone$ which changes the state of requesting vendor to VERIFIED. The vendor can then reply to the customer on either channel $Approval$ (Table 4, lines 25 to 34) or $Disapproval$ (Table 4, lines 36 to 39).

The second task for the vendor is to send touchstone to requesting vendor. A request for touchstone of an ecoin,eid, from vendor, VID, to vendor, $Vtsloc[VID]$, is modeled using channel $ReqToucStone$. This event is enabled when a vendor is in IDLE status. This event stores the details of the touchstone and changes the status to RECVREQ. This enables the reply to the requesting vendor on channel $SendTouchStone$. The status will change to IDLE.

The Netpay process is composed of customer, vendor and broker process. There are three vendors, two customers and one broker in this model. The next section will look at the correctness of this protocol. This model will be used to prove three properties namely chain of trust, preventing double spending and non-blocking behaviour.

V. CORRECTNESS OF THE NETPAY PROTOCOL

The section assumes that we have cooperative customers and vendors who adhere to the protocol. The touchstone and payword are not stored together in the client side Netpay protocol. Validity of an ecoin is dependent on the touchstone. Vendor or Broker stores the touchstone for each payword and the customer stores the payword. The customer will store the touchstone location only. In that case we want to verify that an ecoin remains valid. To prove this, we will show that for any ecoin there exists a chain of touchstone locations that will lead to the broker who issued the ecoin. We will show existence of such a chain, even though it cannot be reconstructed from locally available information. In the Server-side Netpay protocol, the touchstone and payword are stored together at a trusted party so there was no need to prove the

Table 3 Vendor Process

```
Vendor(VID) = [VEN_STATUS[VID] == IDLE &&
    (||x:{0..(VENDORCONST-1)}@
    (VENDORDB[VID][x]==-1))]
    Spend[VID]?cid.eid1.amt.tsloc
        {
            var flag = true;
            Vcid[VID] = cid;
            Veid[VID] = eid1;
            Vamt[VID] = amt;
            if(tsloc != VID)
            {
                VEN_STATUS[VID]=REQTSTONE;
                Vtsloc[VID] = tsloc;

            }else if(tsloc == VID)
            {
                var index = 0;
                while(index < VENDORCONST)
                {
                    if(VENDORDB[VID][index] == eid1)
                    {
                      VEN_STATUS[VID]=VERIFIED;
                      index = VENDORCONST;
                      flag = false;
                    }
                    index = index + 1;
                }
            }
        }->Vendor(VID)
    [][VEN_STATUS[VID] == VERIFICATION]
    SendTouchStone[VID]?vid.eid1
        {
            var index = 0;
            if(eid1 == Veid[VID])
            {
                VEN_STATUS[VID]=VERIFIED;
                while(index < VENDORCONST)
                {
                    if(VENDORDB[VID][index] == -1)
                    {
                      VENDORDB[VID][index] = eid1;
                      index = VENDORCONST;
                    }
                    index = index + 1;
                }
            }
            else
                VEN_STATUS[VID]=IDLE;

        }->Vendor(VID)
    [][VEN_STATUS[VID] == RECVREQ]
    SendTouchStone[Vvid[VID]]!VID.Veid[VID]
        {
            Veid[VID] = -1;
            VEN_STATUS[VID]=IDLE
        }->Vendor(VID)
    [][VEN_STATUS[VID] == REQTSTONE]
    ReqTouchStone[Vtsloc[VID]]!VID.Veid[VID]
        {
            VEN_STATUS[VID]=VERIFICATION;
        }->Vendor(VID)
```

Table 4 Vendor Process (continued)

```
    [][VEN_STATUS[VID] == IDLE]
        ReqTouchStone[VID]?vid.eid1
        {
            Vvid[VID] = vid;
            var index = 0;
            var flag = false;
            while(index < VENDORCONST)
            {
                if (VENDORDB[VID][index] == eid1)
                {
                    flag = true;
                    VENDORDB[VID][index] = -1;
                    index = VENDORCONST;
                }
                index = index + 1;
            }
            if(flag == false)
                Veid[VID] = -1;
            else
                Veid[VID] = eid1;
            VEN_STATUS[VID]=RECVREQ
        }->Vendor(VID)
    [][VEN_STATUS[VID] == VERIFIED &&
    Vamt[VID] == ISPOSITIVE]
    Approval[Vcid[VID]]!VID.Veid[VID].ISPOSITIVE
        {
            VEN_STATUS[VID]=IDLE
        }->Vendor(VID)
    [][VEN_STATUS[VID] == VERIFIED &&
    Vamt[VID] == ISPOSITIVE]
    Approval[Vcid[VID]]!VID.Veid[VID].ISZERO
        {
            VEN_STATUS[VID]=IDLE
        }->Vendor(VID);
    [][VEN_STATUS[VID] == VERIFIED ]
    Disapproval[Vcid[VID]]!VID.Veid[VID]
        {
            VEN_STATUS[VID] = IDLE;
        } ->Vendor(VID)
```

validity of payword. Instead, we proved in [5] that the ecoin will not be lost by vendors.

Furthermore, we show that at most one vendor can have a touchstone of an e-coin. The length of an e-coin, and thus its amount, is abstracted, and we assume that subtracting from the amount is dealt correctly by the trusted vendor. The only remaining way to double spend would be to have touchstone of one ecoin at two different vendors. We show that an e-coin cannot be spent twice at different vendors. Finally, we show that there is a deadlock in the protocol and we will present a solution for this.

A. Chain of Trust

Touchstones are transferred from broker to vendor and from one vendor to another vendor, and the customer keeps track of the location of the touchstone. An ecoin without a touchstone will make that coin invalid. The overall property that we will prove is that the location of the touchstone as recorded by the customer is correct at the end of the transaction.

For the client-side Netpay protocol, the following two properties can be shown to hold:

1) If the customer is in the IDLE or BUYCOIN state, then the customer will have the location of the touchstone

pointing to the vendor with the touchstone.

2) If the customer is in the SPENDING state, then the customer will have the location of the touchstone, or it will point to the vendor which will receive the touchstone after the next exchange.

The following lists the goals defined in the PAT model checker:

- Property 1 shows for each e-coin ID held by a customer, that there exists a corresponding e-coin in the broker database.
- Property 2 shows for each e-coin ID held by a customer, that broker will have a corresponding touchstone if the touchstone location in the $Ewallet$ database is $BROKERID$.
- Property 3 shows for each e-coin ID held by the customer, that if the location of the touchstone is not the broker ID and the customer status is IDLE or BUYCOIN, then there exists a corresponding touchstone at that location.
- Property 4 shows for each e-coin ID held by the customer, that if the location of the touchstone is not equal to the broker ID and the customer status is SPENDING, then there exists a corresponding touchstone at that location or at a location stored in variable *Vtsloc[VID]*. This means that while the customer may have information that is temporarily not valid, the correct location is stored in an auxiliary variable.

The properties 1 - 4 were verified using the PAT model checker.

Property 1 Chain of Trust - Comparison of customer and broker databases

```
#define Chain_of_Trust_1(&&y:{0..(MAXCOINS-1)};x:{0..
  (CUSTOMERS-1)}@(EWALLET[x][y][0]==-1 ||
  (||z:{0..(BROKERDBCONST-1)}
  @(EWALLET[x][y][0] == BROKERDB[z][0])));

#assert Netpay |=[] Chain_of_Trust_1 ;
```

Property 2 Chain of Trust - Comparison of customer and broker databases

```
#define Chain_of_Trust_2(&&y:{0..(MAXCOINS-1)};x:{0..
  (CUSTOMERS-1)}@(EWALLET[x][y][2]!=BROKERID ||
  (||z:{0..(BROKERDBCONST-1)}
   @(EWALLET[x][y][0] == BROKERDB[z][0])));

#assert Netpay |=[] Chain_of_Trust_2 ;
```

Property 3 Chain of Trust - Comparison of customer and broker databases

```
#define Chain_of_Trust_3(&&y:{0..(MAXCOINS-1)};x:{0..
   (CUSTOMERS-1)};z:{0..(VENDORS-1)}
   @(EWALLET[x][y][2]!=z ||CUS_STATUS[x] != IDLE
   ||(||w:{0..(VENDORCONST-1)}@
   (EWALLET[x][y][0] == VENDORDB[z][w])));

#assert Netpay |=[] Chain_of_Trust_3 ;
```

Property 4 Chain of Trust - Comparison of customer and broker databases

```
#define Chain_of_Trust_4(&&y:{0..(MAXCOINS-1)};x:{0..
  (CUSTOMERS-1)};z:{0..(VENDORS-1)}@(EWALLET[x][y][2]
  !=z||CUS_STATUS[x] != SPENDING || Vtsloc[z] == -1
  || Vtsloc[z] == BROKERID||(||w:{0..(VENDORCONST-1)}
   @(EWALLET[x][y][0] == VENDORDB[z][w] ||
   EWALLET[x][y][0] == VENDORDB[Vtsloc[z]][w])));

#assert Netpay |=[] Chain_of_Trust_4 ;
```

B. Double Spending

Double spending means spending an ecoin twice. A payment protocol should prevent customers from double spending. The Client-side Netpay protocol prevents double spending through the use of touchstone and an index. Recall that a touchstone is the root payword in a chain of ecoins and an index is the index of the payword in that chain. Index also indicates the number of hashes to apply on the payword to get a result equal to touchstone. Note, that because we abstract the exact amount of an e-coin it can be spent as long as the amount is ISPOSITIVE. The customer can double spend at two different vendors if the vendors have the touchstone for that ecoin. We prove that for each chain of e-coins there exists a touchstone at a vendor at any time. This is expressed in Property 5: No two vendors have the touchstone for the same e-coin ID. A Windows 8, i7 processor, 3.2 GHz and 6 GB RAM machine took about eight minutes to verify all 5 properties.

Property 5 Double Spending

```
#define DoubleSpending(&&x:{0..VENDORS-1};a:{0..
  VENDORS-1}@(&&y:{0..VENDORCONST-1};z:{0..
  VENDORCONST-1}@((VENDORDB[x][y]==-1||
  VENDORDB[a][z]==-1 || a==x)||
  (VENDORDB[x][y]!=VENDORDB[a][z])));

#assert Netpay |=[] DoubleSpending;
```

C. Non-Blocking Behavior

As described in section IV, processes use channels for communication. A sender and receiver process need to synchronize on a channel. A sender process will not be able to send unless the receiver is ready to receive. This means that the sender process will be blocked if there is no receiver process enabled. If this blocking is indefinite, then the protocol is in a deadlock. There exists a deadlock in the client-side Netpay protocol. This has been verified using the PAT model checker by the property, $\#assert\ Netpay\ deadlockfree;$.

PAT did identify a deadlock in the protocol. Figure 1 depicts the trace that was generated by PAT model checker. This happens when two vendors are in REQSTONE status and there is a circular wait for these vendors to be in IDLE state. The deadlock occurs as follows: Customer(0) buys ecoins with Broker(3) and spends ecoins with ID 0 at vendor(0). Vendor(0) requests and receives touchstone from broker. Vendor(0)

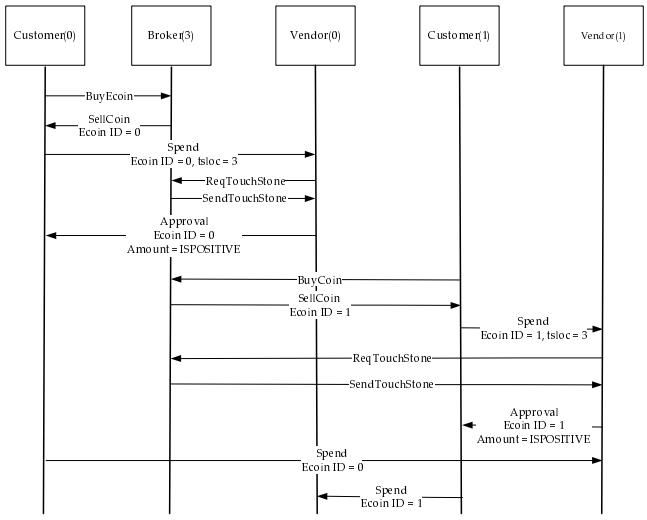

Fig. 1. Deadlock in Netpay protocol

TABLE I
COMPARISON

Properties	Server-Side	Client-Side
Chain of Trust	Valid	Valid
Double Spending	Valid	Valid
Non-Blocking Behaviour	Invalid	Invalid

approves payment for Customer(0) and provides balance for ecoins as ISPOSITIVE. Customer(0) updates the touchstone location with the ID of Vendor(0). Likewise, Customer(1) buys ecoins with ID 1 and spends at Vendor(1). Ecoins with ID 1 is not fully spent. Customer(0) spends ecoins with ID 0 at vendor(1) while Customer(1) spends ecoins with ID 1 at Vendor(0). Vendor(0) has to request the touchstone for ecoin ID 1 from Vendor(1) and Vendor(1) has to request the touchstone for ecoin ID 0 from Vendor(0).

This protocol has been corrected by adding a separate process for the vendor-side to handle reply for touchstone requests. Table 5 illustrates process $VendorP(\text{VID})$ where VID is the vendor ID. Each vendor will now have two processes running: $VendorP(\text{VID})$ and $Vendor(\text{VID})$. The description of the protocol does not specify how many processes a vendor should have [8], although it is presented as if it were a single process.

D. Comparison of Client-Side and Server-Side Netpay

This section compares the client-side and server-side Netpay protocol based on the results of model checking. Server-side Netpay protocol was modeled and verified in [5] where as the client-side Netpay in this. The comparison criteria will be the properties verified for these two variation of the Netpay protocol; chain of trust, double spending and non-blocking behaviour.

The comparison is based on where the customers and vendors are cooperative. In an environment where the customers or vendors are non-cooperative, some of the properties will not

Table 5 Vendor Process

```
VendorP(VID) = [VEN_STATUS[VID] == RECVREQ]
    SendTouchStone[Vvid[VID]]!VID.Veid[VID]{
        Veid1[VID] = -1;
5       VEN_STATUS_P[VID]=IDLE}->VendorP(VID)
    []
    [VEN_STATUS[VID] == IDLE]ReqTouchStone[VID]?
    vid.eid1{
        Vvid[VID] = vid;
10      var index = 0;
        var flag = false;
        while(index < VENDORCONST)
        {
            if (VENDORDB[VID][index] == eid1)
15          {
                flag = true;
                VENDORDB[VID][index] = -1;
                index = VENDORCONST;
            }
20          index = index + 1;
        }
        if(flag == false)
            Veid1[VID] = -1;
        else
25          Veid1[VID] = eid1;
        VEN_STATUS_P[VID]=RECVREQ}->VendorP(VID);
```

hold. Like, currently the protocol caters for if a customer wants to spend twice or an ecoin with zero amount. However, if we would enable a customer to send e-coins that does not exist in the broker, it will cause a deadlock. The current protocol provides no way for a broker to communicate back to vendor that an e-coin does not exist. This problem is, however, easily addressed by adding one more case for a declined payment. Also, if the customer provides the wrong touchstone location, then the vendor has no way to locate the touchstone. This problem can be addressed by adding communication to retrieve the touchstone from the broker and then continue to process the payment normally.

VI. CONCLUSIONS AND FUTURE RESEARCH

This paper modeled the client-side Netpay micropayment system using CSP language. The PAT model checker was used to verify three properties namely chain of trust, preventing double spending and absence of deadlock. The first two properties verified did hold where as the last did not. This shows that there was a deadlock in the protocol. We proposed a solution where a vendor should have two processes; one for customer to spend e-coins and one for replying requesting vendors with touchstone. The deadlock was resolved when the property was verified again.

The verification was performed under the strong assumption that customers and vendors are cooperative. Future research involves proving whether the chain of trust and preventing double spending holds when the customers and vendors are non-cooperative.

REFERENCES

[1] M. Bastiaan. Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin.

http://referaat.cs.utwente.nl/conference/22/paper/7473/preventing-the-51-attack-a-stochastic-analysis-of-two-phase-proof-of-work-in-bitcoin.pdf, 2015.

[2] W. Beukema. Formalising the bitcoin protocol. http://referaat.cs.utwente.nl/conference/21/paper/7450/formalising-the-bitcoin-protocol.pdf, 2014.

[3] Y. Cai, J. Grundy, J. Hosking, and X. Dai. Software Architecture Modeling and Performance Analysis with Argo/MTE. In *SEKE 2004*, 1990.

[4] K. Chaudhary and X. Dai. P2P-NetPay: An off-line Micro-payment System for Content Sharing in P2P-Networks. *JETWI*, 1(1):46–54, August 2009.

[5] K. Chaudhary and A. Fehnker. Model checking a server-side micro payment protocol. In *Formal Methods for Industrial Critical Systems - 20th International Workshop, FMICS 2015, Oslo, Norway, June 22-23, 2015 Proceedings*, pages 96–110, 2015.

[6] X. Dai and J. Grundy. Architecture for a Component-Based, Plug-in Micro-payment System. In *APWeb2003*, pages 251–262. LNCS, Springer, April 2003.

[7] X. Dai and J. Grundy. Three Kinds of E-wallets for a NetPay Micro-payment System. In *WISE 2004*. LNCS 3306, 2004.

[8] X. Dai and B. Lo. NetPay - An Efficient Protocol for Micropayments on the WWW. In *AusWeb 99*, Australia, 1999.

[9] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The Millicent Protocol for Inexpensive Electronic Commerce. In *www95*, December 1995.

[10] R. Hauser, M. Steiner, and M. Waidner. Micro-payments Based on ikp. In *SECURICOM 96*. LNCS, 1996.

[11] A. Herzberg and H. Yochai. Mini-pay: Charging Per Click on the Web. 1996.

[12] C. A. R. Hoare. Communicating Sequential Processes. Prentice Hall, 1985.

[13] N. Nisan, S. London, O. Regev, and N. Camiel. Globally Distributed Computation Over the Internet. The POPCORN project. In *ICDCS'98*. IEEE, 1998.

[14] R. Rivest and A. Shamir. PayWord and MicroMint: Two Simple Micropayment Schemes. pages 307–314. LNCS, 1998.

[15] J. Sun, Y. Liu, and J. Dong. Protocol Analysis Toolkit. http://www.comp.nus.edu.sg/ pat/.

[16] K. Wei, A. Smith, Y. Chen, and B. Vo. WhoPay : A Scalable and Anonymous Payment System for Peer-to-Peer Environments. In *Distributed Computing Systems*. IEEE, 2006.

[17] B. Yang and H. Garcia-Molina. PPay: Micro-payments for Peer-to-Peer Systems. In *CSS 2003*, pages 300–310, 2003.

[18] E. Zou, T. Si, L. Huang, and Y. Dai. A New Micro-payment Protocol Based on P2P Networks. In *ICEBE'05*, 2005.