International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:6, No:12, 2012

Modeling and Verification for the Micropayment Protocol Netpay

Kaylash Chaudhary, and Ansgar Fehnker

Abstract**—**There are many virtual payment systems available to conduct micropayments. It is essential that the protocols satisfy the highest standards of correctness. This paper examines the Netpay Protocol [3], provide its formalization as automata model, and prove two important correctness properties, namely absence of deadlock and validity of an ecoin during the execution of the protocol. This paper assumes a cooperative customer and will prove that the protocol is executing according to its description.

Keywords**—**Model, Verification, Micropayment.

I. Introduction

SYSTEM verification techniques' are nowadays applied to the design of many ICT systems. Correctness is particularly important for payments systems, such as Netpay. Netpay is a virtual payment protocol for small purchases, typically on the internet. With the increase of paid services and content on the internet, these online payment system promise the ease of using cash. There are alternatives to credit card based systems, which cannot be used due to their high cost per transaction. Mircopayment systems for online payments that have been proposed in recent years are Netpay as described in [1],[2] and [3], Millicent [4], Micro-mint [5], Payword [5], MiniPay [6], Micro-iKP [7] and POPCORN [8].

This paper models the Netpay protocol [1] using automata, in particular the syntax of interface automata, which distinguishes nicely between inputs and outputs. We will use this model to show two essential properties: First, that the different components cannot block actions of other components indefinitely. This is a fairness type property, which ensures that the protocol will progress. Second, we will show that electronic money called ecoins, will remain valid throughout, i.e. that there exists a chain of trust that links any ecoin to a trusted broker, even if, for the sake of anonymity, this chain is not stored explicitly by the protocol. The model in this paper covers the handling of ecoins, and omits the parts of the protocol that are concerned with spend redeeming ecoins. For the properties under consideration, it was not necessary to include these parts.

The next section of this paper will introduce automata model that we will use for modeling and verification. Section III will give an overview of the Netpay protocol, while Section IV gives the detailed model. Section V formalizes the properties and will give the invariant proof. Section VI concludes the paper with a discussion of future work.

Kaylash Chaudhary is an Assistant Lecturer in Computing Science at the University of the South Pacific. He is currently pursuing a PhD Degree in the School of Computing, Information & Mathematical Sciences, University of the South Pacific, Suva, Fiji (e-mail: chaudhary_k@usp.ac.fj).

Ansgar Fehnker is a Professor in Computing Science at the University of the South Pacific.

II. Interface Automata

This paper will use a model based on interface automata to model the Netpay protocol. Interface automata are in particular suitable because they provide a clean distinction between input and output actions. They are similar to input/output automata as defined in [9], however do not require that all input actions are always enabled. This paper will model the each player in the protocol -broker, vendor and customer -as an interface automaton, such that the entire system is defined by the composition thereof.

We define interface automaton as a tuple $A = (S, I, H, O, S^{init}, R)$ with the following components:

1. A finite set of states *S*.
2. Finite sets of input actions *I*, hidden or internal actions *H*, and output actions *O*. Input, hidden and output actions are disjoint sets.
3. A set of initial states S^{init}, a subset of S.
4. A transition relation *R*, a subset of $S \times (input \cup output) \times S$ where $(I \cup H \cup O)$ is the set of actions.

The model distinguishes between input, hidden, and output actions. It is assumed that output actions are controlled by the system whereas the input actions are controlled by the environment. An execution fragment is an alternating finite or infinite sequence of states and actions $s_0, a_0, s_1, \ldots\ldots$ with $(s_i, a_i, s_{i+1}) \in R$. An execution fragment starting with a start state $s_0 \in s^{init}$ is known as an execution. The final state of a finite execution is identified as reachable state.

Interface automata can only be composed if the input and output actions match, this means that the input actions are disjoint, and that the output actions are disjoint, and hidden actions are disjoint from any action of the other automaton. Two interface automata $A_1 = (S_1, I_1, H_1, O_1, S_1^{init}, R_1)$ and $A_2 = (S_2, I_2, H_2, O_2, S_2^{init}, R_2)$ are said to be *composable* if $I_1 \cap I_2 = \emptyset, O_1 \cap O_2 = \emptyset,\ H_1 \cap (I_2 \cup H_2 \cup O_2) = \emptyset$, and $(I_1 \cup H_1 \cup O_1) \cap H_2 = \emptyset$. The set of shared input/output actions $shared(A_1, A_2)$ is defined as $(I_1 \cap O_2) \cup (O_1 \cap I_2)$. If two interface automata A_1 and A_2 are composable their composition is defined as a interface automaton $A' = (S', I', H', O', S^{init'}, R')$ with

- $S' = S_1 \times S_2$.
- $I' = (I_1 \cup I_2) \backslash shared(A_1, A_2)$.
- $I' = (O_1 \cup O_2) \backslash shared(A_1, A_2)$.
- $H' = H_1 \cup H_2 \cup shared(A_1, A_2)$.

- $S^{init'} = S_1^{init} \times S_2^{init}$.
- $R' =$
$\left\{((v,u),a,(v',u)) \middle| \begin{matrix} a \notin shared(A_1,A_2) \wedge \\ (v,a,v') \in R_1 \end{matrix} \right\} \cup$
$\left\{((v,u),a,(v,u')) \middle| \begin{matrix} a \notin shared(A_1,A_2) \wedge \\ (u,a,u') \in R_2 \end{matrix} \right\} \cup$
$\{((v',u),a,(v',u')) | a \in shared(A_1,A_2) \wedge$
$(v,a,v') \in R_1 \wedge (u,a,u') \in R_2\}$

This means that if an input and output action is shared they will have to synchronize. This means also that an output action might be blocked in a state, if there is no corresponding outgoing input action. Input/output automata resolve this issue by requiring all input actions to be enabled in all states. Interface automata in contrast have the notion of an error state.

An error state is a state in which one component of the compositions has an outgoing output action, while the other component does not have the matching input action.

The model presented departs from interface automata in one important aspect, namely that we do accept error states, rather than remove them from the set of states. We alternatively adopt a notion of (weak) fairness for output actions. Weak fairness means that an action cannot be continuously enabled and not be taken. In the context of interface automata we require a modified notion. We require that an output action cannot be continuously enabled, while the corresponding input action is continuously disabled. This means that an output action might be temporarily blocked, but not indefinitely. This is a property that needs to be proven for the Netpay model.

The model of the Netpay protocol will use a precondition effect style of specification, with variables, and parameterized actions. The state of an automaton is defined by the values assigned to all variables. Preconditions are used to define sets of states in which actions with a common label are enabled while the effect is used to define the successor states. The precondition are omitted is true. Transitions can synchronize if they have the same parameterized action label this means the state change in both automata will take place as a single atomic step.

III. The Netpay Protocol

This section describes the Netpay protocol for micro-payments proposed by Dai [2]. The general structure of the Netpay protocol is depicted in Figure 1. The Netpay micropayment system comprises three different types of actors: customers, brokers and vendors. It is assumed that the broker is honest and trusted by both customers and vendors. The brokers' key responsibility is to register customers, credit the account of vendors and debit customer account. The payments occur between customers and vendors.

A key idea of Netpay is the use of cryptographic hash functions such as MD5 or SHA2. The description of Netpay refers to a number of micropayment and cryptographic terminologies such as:

- **One way hash function**. Netpay as proposed by Dai [2] uses the MD5 algorithm as one way hash function. Electronic coins are generated and verified using this hash function.
- **Payword**. A payword has a length of 32 digit hexadecimal, and is generated with the one-way hash function.
- **Payword Chain**. A payword chain is a series of paywords, generated from the same seed. For example, a payword chain of length 10 might represent 10 cents.
- **E-wallet**. An e-wallet is a database for storing one or more payword e-coins.
- **Seed**. A randomly selected value used to generate payword chains.
- **Touchstone**. In addition to the payword chains, vendors and broker store the touchstone. The touchstone is used to verify the e-coins sent by the customer.
- **Index**. Vendors also store an index for each payword chain, which indicates the current spent amount of each payword chain.

Let $\mathcal{S}$ be a set of seeds, and $\mathcal{P}$ be a set of paywords. A payword chain is then a finite series of payword from $\mathcal{P}^*$. Given a one way hash function h, a payword chain $W_0, \ldots \ldots, W_n$ is created from a seed W_{n+1} by applying the hash function $n + 1$ times to the seed. This means $W_i = h(W_{I+1})$, for $i = 0,\ldots, n$. The touchstone is kept separately, to verify the payword chain. Given the touchstone, a payword chain $W_1, \ldots, W_m$can be verified by applying the hash function to the elements of the payword chain, and check that it is indeed a chain, and that the hash of the first element is equal to the touchtone.

We define the following two functions to be used by the model. First a function *createPW*:$\mathcal{S} \times \mathcal{N} \to \mathcal{P} \times \mathcal{P}^*$, which returns for a given seed s and an amount n a triplet of the touchstone, a payword chain and the seed.

$$createPW(s,n) \triangleq \left(h^{(n+1)}(s), (h^n(s), \ldots, h^1(s)\right), s)$$

where $h^i(s)$ is the i-th application of the hash function to a seed value. Then a function *verifyPW*:$\mathcal{P} \times \mathcal{P}^* \to \mathbb{B}$ which returns for a touchstone $W_0$and a series of paywords $(W_1, \ldots, W_n)$ the Boolean value true if it is a legitimate payword chain:

$$verifyPW(W_0(W_1, \ldots, W_n)) \triangleq \bigwedge_{i=1,\ldots,n} W_{i-1} = h(W_i)$$

The protocol contains four basic types on transactions: Customer-Broker, Customer-Vendor, Vendor-Vendor and Vendor-Broker transactions. These are illustrated in Figure 1. We assume that every customer has a unique ID from a set $\mathcal{C}$, while every broker or vendor has a unique ID from the set $\mathcal{V}$. Note, that vendors and broker share the same ID space. Every newly created payword chain will be associated with a unique e-coin ID from the set $\mathcal{E}$. An e-coin is defined as triplet $(eid, pwords, vid)$of an e-coin ID $eid \in \mathcal{E}$, a payword chain $pwords \in \mathcal{P}^*$, and a vendor/broker ID $vid \in \mathcal{V}$. The latter denotes which vendor or broker holds the touchstone.

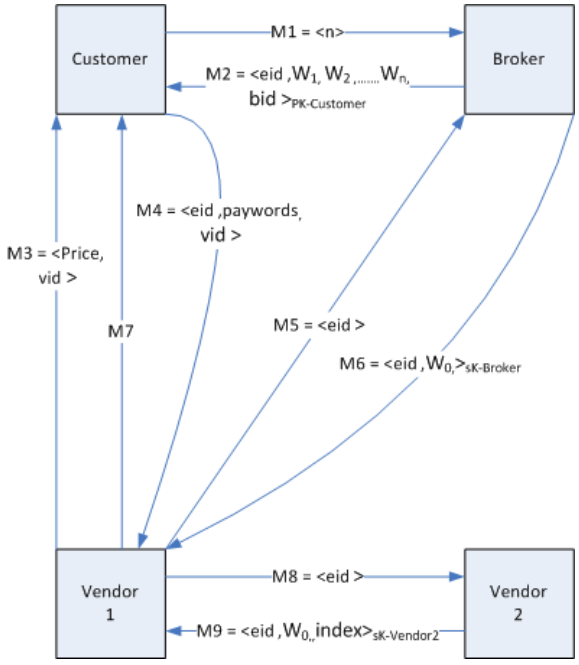

Fig. 1 A representation of Netpay protocol

The four types of transaction can be characterized as follows:

A. Customer-Broker Transaction

The customer initiates the transaction in the Netpay protocol by registering and sending an integer nto a broker, where nis the amount of paywords requested. The Broker generates the payword chain of length n and assigns to that chain a unique e-coin ID, which together with the broker ID will constitute a new e-coin. The broker encrypts the e-coin with the customer's public key (M2, Figure 1) and sends it to the customer. The customer decrypts it and stores in its e-wallet.

B. Customer-Vendor Transaction

The vendor sends the cost and its ID (which is the host/port number) to the customer when the customer wishes to buy goods. The customer compares the ID with the vendor ID of the e-coins in its e-wallet. If there is an e-coin with a matching vendor ID and the length of the payword chain is greater than the cost, the customer sends the e-coin (M4). If there is no e-coin with a matching vendor ID, the customer sends any e-coin with a sufficiently long payword chain (M4). Note, that the latter is just an optimization, and not require for correctness.

The vendor ID contained in an e-coin refers to the vendor where this e-coin chain was spent last, or to the broker who generated the e-coin, if the e-coin has not been spent before. The vendor verifies e-coins by requesting the touchstone from this broker (M5) or vendor (M8). The response from the broker or vendor is encrypted by the secret key. If the verification is successful, the vendor sends the required information to the customer.

C. Vendor-Vendor Transaction

This transaction occurs when one vendor requests a touchstone and index from another vendor for a particular e-coin ID (M8 and M9).

D. Vendor-Broker Transaction

To redeem spent e-coins the vendor sends the touchstone, customer ID, vendor ID, payword chain and payment to the broker for each e-coin spent with the vendor. The broker will verify each e-coin received from the vendor by performing hashes on it and will count the amount of the paywords. The broker will credit the corresponding amount to the vendors account if all paywords are valid. This paper focuses on the spending of e-coins, and omits redemption of e-coins from the model.

The Netpay protocol specifies three kinds of e-wallets, depending on whether it is a client-side, server-side or cookie-based e-wallet. In this paper we will model the protocol for a client side e-wallet.

E. Properties of Netpay Protocol

The properties of the Netpay micro-payment system are defined in [10] as follows:

- Security: the aim of security is to prevent any party from cheating the system. For example, double spending of coins and creation of false coins.
- Anonymity: the customer anonymity should be protected. A fundamental property of physical cash is that the relationship between customers and their purchases is untraceable. Anonymity as provided by Netpay can be proven by anonymous simulation which is introduced in [11].
- Robustness: the protocol is tolerant of network bottlenecks and broker/authorizer down-time. The broker will be only involved in the generation of e-coins and providing touchstone for the first set of e-coins. If the broker is down, the protocol should be able to operate for customers who would like to spend a partially spent payword chain with a vendor.

This paper will consider two properties that are important for customers that use the protocol correctly. The first is that any valid e-coin remains valid, even if it is partially spend. In particular, the protocol should guarantees that there exists for every e-coin a chain of trust back to the broker, even though the system does not store this chain explicitly, to protect anonymity. The second property is that the protocol does not block any participant in the protocol indefinitely.

IV. Description of Netpay Protocol using IA

This section gives a detailed description of the Netpay protocol as interface automata. We use one automaton each for customers, vendors, and brokers. The overall system is described by the composition of all the above automata.

In the following we use the following conventions: Constants will be written with capital letters, local variable are written with a capital initial letter, and parameters and free variables with lower case letters only. Sets will be denoted

mostly by calligraphic letters. We assume that sets for IDs and paywords have a distinguished "undefined" element.

A. IA for Customer

The automaton $Customer(CID)$ modeling a customer automaton is shown in Table I. Every customer has a unique identifier CID, which is a parameter of the automaton. The customer will have an e-wallet, which is a partial function $Ewallet \subseteq \mathcal{E} \times \mathcal{P}^* \times \mathcal{V}$, which associates an e-coin ID with a payword chain and a vendor/broker ID. The e-wallet is initially empty. There are three possible control locations in the cus-tomer automaton which are $IDLE$, $BUYECOIN$ and $BUYGOODS$. In addition the automaton has local variable $BVid$ to store the ID of the vendor who received the ecoin, or the ID of the broker who was asked to supply an ecoin.

We will now describe the automaton's actions. The model assumes the customer is already registered and has started using the protocol. In order to buy e-coins the customer has an output action **Send**(CID, n, bid) where CID is the customer ID and $n \in \mathbb{N}$ the amount requested, and $bid \in \mathcal{V}$ the ID of the broker. The customer automaton will now be in $BUYECOIN$ state, and the broker ID stored in variable $BVid$.

If the automaton is in location $BUYECOIN$, input action **SendEcoin**$(ecoin, CID, BVid)$ models the sending of $ecoin$ from customer CID to broker $BVid$. The effect of this input action adds the ecoin to the $Ewallet$ and changes the status from $BUYECOIN$ to $IDLE$.

Buying goods from a vendor is also done in two steps. The first is modeled by action **SendPaywords**$(CID, newcoin, vid)$. The precondition is that the customer has an ecoin in the ewallet. The length or size of an ecoin $|ecoin|$is defined as the length of the payword chain. The parameter $newcoin$ is a new ecoin, which includes the first n paywords of the payword chain.

TABLE I
CUSTOMER AUTOMATON

$Customer(CID)$

State:
$Ewallet \subseteq \mathcal{E} \times \mathcal{P}^* \times \mathcal{V}$, *initially* $\emptyset$
$Status \in \{IDLE, BUYECOIN, BUYGOODS\}$, *initially* $IDLE$
$BVid \in \mathcal{V}$, *initially undefined*
$Newecoin \in \mathcal{E} \times \mathcal{P}^* \times \mathcal{V}$, *initially undefined*
$Oldecoin \in \mathcal{E} \times \mathcal{P}^* \times \mathcal{V}$, *initially undefined*

Input Actions:

SendEcoin$(ecoin, CID, BVid)$
Pre: $Status = BUYECOIN$
Effect: $Ewallet' = Ewallet \cup ecoin$
$Status' = IDLE$

SendInformationBought$(CID, BVid)$
Pre: $Status = BUYGOODS$
Effect: $Status' = IDLE$
$Ewallet' = (Ewallet \backslash \{Oldecoin\}) \cup \{Newecoin\}$

Output Actions:

Send(CID, n, bid)
Pre: $Status = IDLE$
Effect: $Status' = BUYECOIN$
$BVid' = bid$

SendPaywords$(CID, newcoin, vid)$
Pre: $Status = IDLE$
$\exists ecoin \in Ewallet, n \leq |ecoin| s.t. newecoin = charge(ecoin, n)$
Effect: $Status' = BUYGOODS$
$BVid' = vid$
$Oldecoin' = ecoin$
$Newecoin' = remainder(ecoin, n, vid)$

This new ecoin is computed by the function $charge(ecoin, n)$ which maps an e-coin $(eid, (W_1, \ldots, W_m), vid)$ to $(eid, (W_1, \ldots, W_n), vid)$ if $n \leq m$. The amount n is chosen in this model non-deterministically; we omitted the part of the protocols that checks if the funds are sufficient from the protocol. In the context of this paper we are interested only if the ecoin remains valid.

The effect of this action is that we change to status $BUYGOODS$, store the vendor ID in $BVid$, and store the ecoin selected from the ewallet in $oldecoin$, and the new ecoin, i.e. the remaining paywords that were not sent to the vendor, in $Newcoin$. The function $remainder(ecoin, n, vid)$ is defined to map $(eid, (W_1, \ldots, W_m), vid)$ to $(eid, (W_{n+1}, \ldots, W_m), vid)$ if $n \leq m$.

In status $BUYGOODS$ the customer waits for confirmation. The reply from the vendor is modeled by input action **SendInformationBought**$(CID, Bvid)$. It is enabled if the parameter vid, which will be the replying vendor ID, is equal

to the stored ID, the vendor the e-coin was sent to. It will replace the old ecoin in the ewallet with the new ecoin.

B. IA for Brokers

The behavior of the broker will be modeled by an automaton $Broker(BID)$. The broker ID BID is from the same set $\mathcal{V}$ as the vendor IDs. This is because to vendors a broker behaves like a vendor.

TABLE II
BROKER AUTOMATON

$Broker(BID)$

State:
$BrokerDB \subseteq \mathcal{E} \times \mathcal{P}, initially\ \emptyset$

$Status \in \{IDLE, WAIT, GENERATEECOIN\}$, $initially\ IDLE$
$Eid \in \mathcal{E}, initially\ undefined$
$Cid \in \mathcal{C}, initially\ undefined$
$Seed \in \mathcal{S}\ initially\ undefined$
$Tstone \in \mathcal{P}, initially\ undefined$
$Pwords \in \mathcal{P}^*, initially\ empty$
$BVid \in \mathcal{V}, initially\ undefined$

Input Actions:

Send(cid, n, BID)
Pre: $Status = IDLE$
Effect: $Status' = GENERATEECOIN$
$Cid' = cid$
$Eid' = new(\mathcal{E})(Tstone', Pwords', Seed') = createPW(n, new(\mathcal{S}))$

GetTouchStone(eid, vid, BID)
Pre: $Status = IDLE$
$(\exists (eid, ts) \in BrokerDB)$
Effect: $Status' = WAIT$
$Eid' = eid$
$Tston' = ts$
$BVid' = vid$

Output Actions:

SendEcoin$((Eid, Pwords, BID), Cid, BID)$
Pre: $Status = GENERATEECOIN$
Effect: $Status' = IDLE$
$BrokerDB' = BrokerDB \cup (Eid, Tstone)$

SendTouchStone$(Eid, 1, Tstone, BID, BVid)$
Pre: $Status = WAIT$
Effect: $Status' = IDLE$

The broker keeps information on generated e-coins in a database $BrokerDB \subseteq \mathcal{E} \times \mathcal{P}$. An entry (eid, cid, ts, s, n) captures the e-coin ID eid, the customer ID cid, the touchstone ts, the seed s, and the amount n. Other local variables are $Eid, Cid, Seed, Tstone, Pwords,$ and $BVid$, which are used to store intermediate results, when generating an ecoin, or replying to a request to send the touchstone for an ecoin.

The broker automaton models two possible exchanges of messages; the first when a customer requests a new ecoin, and the second when a vendor asks to get the touchstone that belongs to a given ecoin.

The first action of the broker automaton models a request to generate e-coins. Input action **Send**(cid, n, BID) has a parameter cid, the ID of the requesting customer, n the amount requested, and the ID of the broker itself. The input action is enabled when the status is $IDLE$, and it will change to $GENERATEECOIN$. This action will also store the ID of the requesting customer in Cid. Finally it will compute a new ecoin ID, and a triplet of a touchstone, payword chain and a seed. The function newselects a new value from either $\mathcal{S}$or $\mathcal{E}$, i.e. it will select a new seed and ecoin ID. This input action will be followed by the output action **SendEcoin**$((Eid, Pwords, BID), Cid, BID)$. It models the sending of a new ecoin, with ID Eid, payword chain $Pwords$ and vendor/broker ID BID to customer Cid. This action is enabled if the status is $GENERATEECOIN$. The status will change to $IDLE$ and the necessary information on the ecoin will be stored in the broker database $BrokerDB$.

The request by a vendor for a touchstone is modeled by input action **GetTouchStone**(eid, vid, BID). It has as parameters the ID eid of the ecoin that needs to be verified, the ID vid of the vendor making the request, and ID of the broker itself. This action is enabled if the status is $IDLE$ and if an ecoin with ID eid exists in the broker database $BrokerDB$. This action will change the status to $WAIT$, and store the ecoin ID eid and the touchstone ts from the broker database, and also store the ID of the requesting vendor vid.

The reply to the touchstone request is modeled by output action **SendTouchStone**$(Eid, 1, Tstone, BID, BVid)$, which models sending the ecoin ID Eid, the index 1, the touchstone $Tstone$, from the broker BID to the vendor $BVid$. The index has constant value 1, because this reply is sent by the broker, when the ecoin is still fresh. The action is enabled if the status is $WAIT$, and it change the status to $IDLE$.

C. IA for Vendor

The automaton $Vendor(VID)$ modeling a vendor automaton is shown in Table III and Table IV. Since there will be many vendors, we use the vendor ID VID. Each vendor will maintain a vendor database $VendorDB \subseteq \mathcal{E} \times \mathcal{P}$, a partial function from a ecoin ID to a touchstone. Recall that the touchstone of an ecoin with n paywords, is obtained by applying the one-way-hash function $n + 1$ times to the seed.

TABLE III
VENDOR AUTOMATON (INPUT ACTIONS)

$Vendor(VID)$

State:
$VendorDB \subseteq \mathcal{E} \times \mathcal{P}$
$Status \in \{IDLE, WAIT, SEARCH, VERIFICATION, VERIFIED\}, initially IDLE$

$Eid \in \mathcal{E}, initially undefined$
$BVid \in \mathcal{V}, initially empty$
$Pwords \in \mathcal{P}^*, initially empty$
$Tstone \in \mathcal{P}, initially undefined$
$Cid \in \mathcal{C}, initially empty$

Input Actions:

SendPaywords$(cid, (eid, pwords, tloc), VID)$
Pre: $Status = IDLE$
Effect: $if\,(tloc! = VID)$
{
$Status' = WAIT$
$Cid' = cid$
$Eid' = eid$
$Pwords' = pwords$
$BVid' = tloc$
}
$elseif\,(verifyPW(VendorDB(eid), pwords))$
{
$Status' = VERIFIED$
$Cid' = cid$
$Eid' = eid$
$Pwords' = pwords$
$BVid' = tloc$
}
$else$
{
$Status' = IDLE$
}

SendTouchStone$(eid, ts, BVid, VID)$
Pre: $Status = VERIFICATION$
Effect: $if\,(verifyPW(ts, Pwords))$
{
$Status' = VERIFIED$
}
$else$
{
$Status' = IDLE$
}

GetTouchStone$(eid, BVid, VID)$
Pre: $Status = IDLE$
$(\exists (eid, ts) \in VendorDB)$
Effect: $Status' = SEARCH$
$Tstone' = ts$
$Eid' = eid$

TABLE IV
VENDOR AUTOMATON (OUTPUT ACTIONS)

Output Actions:

GetTouchStone$(Eid, BVid, VID)$
Pre: $Status = WAIT$
Effect: $Status' = VERIFICATION$

SendInformationBought(cid, VID)
Pre: $Status = VERIFIED$
Effect: $Status' = IDLE$

$VendorDB' = (VendorDB \backslash \{(Eid, VendorDB(Eid))\}) \cup \{(Eid, Pwords_{last})\}$

SendTouchStone$(Eid, Tstone, BVid, VID)$
Pre: $Status = SEARCH$
Effect: $Status' = IDLE$

The status of the vendor can be either $IDLE, WAIT, SEARCH, VERIFICATION$ or $VERIFIED$. Furthermore the vendor uses local variables $Eid, BVid, Pwords, Tstone$ and Cid to store information about the customers and vendors/broker it communicates with, and the ecoins that need to be verified.

The vendor performs two major tasks: verifying e-coins received from customers and providing the touchstone to a requesting vendor. Verification of ecoins has to consider two cases: that the touchstone for the ecoin is with the vendor, or that the touchstone is with another vendor/broker.

The first task will be initiated upon receipt of the input action **SendPaywords**$(cid, (eid, pwords, tloc), VID)$. This action models the sending of an ecoin $(eid, pwords, tloc)$from customer cid to the vendor VID. This action is enabled if the status is $IDLE$.

If the location of the touchstone $tloc$ is not equal to the vendor ID VID it will change its status to $WAIT$and store the customer ID cid and the ecoin ID eid, the payword chain $pwords$, and the touchstone location $tloc$.

If the location of the touchstone $tloc$ is equal to the vendor ID VID, it will use the stored touchstone $VendorDB(eid)$to verify the payword chain. If the payword chain can be verified, i.e. $verifyPW(VendorDB(eid), pwords)$is true, then the status changes to $VERIFIED$. The effect stores the customer ID cid, and the ecoin ID eid, the payword chain $pwords$, and the touchstone location $tloc$.

Recall that the customer sends a prefix of a longer payword chain, and that the last payword of that prefix, will be the touchstone for the remaining payword chain, which is still with the customer. Note, that this differs from the description by Dai [2]. In that paper indices in combination with the original touchstone are used to mark the last payword that has been spent. For the scope of this paper both approaches are equal, however adding indices would add complexity.

If the ecoin cannot be verified by the vendor itself, i.e. if $tloc! = VID$ and the status is $WAIT$, it will request the

touchstone from the vendor/broker. This means any vendor can request a touchstone or reply to such a request.

First, consider that the vendor requests a touchstone. This request is modeled by output action **GetTouchStone**$(Eid, BVid, VID)$. The parameters of this action are the ecoin ID, the ID of the vendor/broker that has the touchstone, and the ID of the vendor itself. The effect is that the vendor will change its status to $VERIFICATION$. The received reply is modeled by input action **SendTouchStone**$(eid, ts, BVid, VID)$, which will in its effect verify the payword chain with respect to the received touchstone, and change its status to $VERIFIED$. If the status is $VERIFIED$ a vendor can inform the customer of the successful transaction which is modeled by output action **SendInformationBought**(cid, VID). This action will update the vendor database, by removing the touchstone and replacing it by the last payword of the stored payword chain.

A vendor who receives a touchstone request will do the following. An incoming request is modeled by input action **GetTouchStone**$(eid, BVid, VID)$, which is similar to the input action with the same name for the broker in Table II. The response to the request is modeled by output action **SendTouchStone**$(Eid, Tstone, BVid, VID)$, after which the vendor is back to status $IDLE$. This action is also similar to output action with the same name for the broker in Table II

In the next section we will look at the correctness of this protocol, assuming a cooperative customer and vendor. In this case the protocol should ensure that for every ecoin there exists a chain of trust back to the issuing broker.

V. Correctness of Netpay Protocol

Given the model of the Netpay protocol in section IV, we prove the correctness of this in this section. The assumption is that we have cooperative customers and vendors who adhere to the protocol. In that case we require that an ecoin remain valid, and will be correctly verified as a valid ecoin. We will in particular show that for any ecoin there exists a chain of touchstones back to the vendors who issued the ecoin. We will show existence of such a chain, even though it cannot be reconstructed from locally available information.

Furthermore we want the protocol to be responsive without any deadlocks. In particular it should be the case that if an output action satisfies the precondition locally, it cannot be blocked indefinitely. This is of course based on the premise that all executions are fair, i.e. that all participants execute their enabled actions eventually.

A. Chain of Trust

The correctness proof will rely on invariants, i.e. properties that can be shown to hold in every state, regardless of state changes. The overall property we prove is that for any ecoin there exists vendor with a touchstone ts, that the ecoin is valid with respect to that touchstone, and that there exists a broker and a positive integer i such that $h^i(ts)$ is the touchstone stored by the broker. The proof is broken into three parts:

1) If a customer receives a new ecoin from a broker, that broker will have a corresponding touchstone.
2) If a customer keeps the remainder of an ecoin after payment to a vendor, then that vendor will have a corresponding touchstone.
3) For any touchstone that a vendor keeps, there exists a corresponding touchstone at a broker.

The proof for the overall property follows from this. The following will describe each property in detail.

Lemma 1: Let $(eid, pwords, bid) \in Ewallet$ of a customer $Customer(CID)$, and let bid be the ID of a broker $Broker(bid)$. Then there exists an entry (eid, ts) in the database $BrokerDB$ of $Broker(bid)$, such that $verifyPW(ts, pwords)$ holds.

Proof: The claim of Lemma 1 is an invariant, and can be proven by induction on the length of execution. The base case of induction is to prove that invariant is true in the initial state. Initially, $Ewallet$ is empty, hence the invariant is true.

Next we have to show that the invariant remains true under all possible actions. We assume that the invariant is true in a predecessor state of an action and show that it will be true in the successor state.

The customer has only two actions that modify the ewallet: **SendEcoin**$(ecoin, CID, bid)$ and **SendInformationBought**$(CID, BVid)$.

- **SendEcoin**$(ecoin, CID, BVid)$
  This action is enabled if the status is $BUYECOIN$, which mean this action was preceded by output action **Send**(CID, n, bid). Both of these actions synchronize with corresponding actions of the broker. The combined action **Send**(CID, n, BID) will have as effect that the broker stored a valid ecoin and its touchstone in its local variables $Eid, Pwords, Tstone,$ i.e. $verifyPW(Tstone, Pwords)$ holds by construction. This step will be succeeded by a combined action **SendEcoin**$((Eid, Pwords, BID), CID, BID)$.
  On the customer side it will store the ecoin $(Eid, Pwords, BID)$ in the ewallet, and on the broker side $(Eid, Tstone)$ in database $BrokerDB$.
- **SendInformationBought**$(CID, BVid)$
  This input action of a customer synchronizes with a corresponding vendor action. It is enabled if the status is $BUYGOODS$, which means that this action was preceded by action **SendPaywords**$(CID, newcoin, vid)$. This action created a new ecoin that replaced the broker ID with a vendor ID. This means that the assumption of Lemma 1, namely that the touchstone location is a broker ID, no longer applies, and thus that the invariant holds.

All other actions of the other automata do not change the ewallet or broker database, and thus have no effect on the invariant of Lemma 1.

The second invariant shows that if an e-coin is in an ewallet and the touchstone location is said to be at a vendor, then the vendor has the touchstone.

Lemma 2: Let $(eid, pwords, vid) \in Ewallet$ of a customer $Customer(CID)$, and let vid be the ID of a vendor $Vendor(vid)$. Then there exists an entry (eid, ts) in the database $VendorDB$ of $Vendor(vid)$, such that $verifyPW(ts, pwords)$ holds.

Proof: This lemma defines as the pervious lemma as an invariant. The claim is true in the initial state, as $Ewallet$ is initially empty.

The customer has only two actions that modify the ewallet **SendEcoin**$(ecoin, CID, bid)$ and **SendInformationBought**$(CID, BVid)$.

- **SendEcoin**$(ecoin, CID, bid)$
  This action will add an ecoin $(eid, pwords, bid)$ in which bid is a broker ID. Hence, the premise of Lemma 2 is false, which makes the invariant true.
- **SendInformationBought**$(CID, BVid)$
  This input action of a customer synchronizes with a corresponding vendor action. It is enabled if the status is $BUYGOODS$, which means that this action was preceded by action **SendPaywords**$(CID, newcoin, vid)$. The effect of the combined action **SendPaywords**$(CID, newcoin, VID)$ is that the vendor stores the first n paywords in the vendors local variable $Pwords$, and the remainder in the customers variable $Pwords$. For clarification we denote the first as $Pwords^C$ and the latter as $Pwords^v$. When customer and vendor successively synchronize on action **SendInformationBought**(CID, VID) the vendor stores the last payword of $Pwords^V$ chain as touchstone in the $VendorDB$, and all successive paywords, $Pwords^C$, as part of the new ecoin in the ewallet of the customer. Since both derive from a valid payword chain, we have that $verifyPW(Pwords\ ^v_{last}, Pwords^C)$ holds.

All other actions of the other automata do not change the ewallet or vendor database, and thus have no effect on the invariant of Lemma 2.

Lemma 3: Let $(eid, ts) \in VendorDB$ of a vendor $Vendor(VID)$, then there exists a broker $Broker(BID)$ and an $n \in \mathbb{N}$ such that for the broker database holds $h^n(ts) = BrokerDB(eid)$.

Proof: This invariant states that a touchstone kept by a vendor is a predecessor in the payword chain. The claim is true in the initial state, as the vendor database is still empty.

The only action that changes the vendor database is **SendInformationBought**$(CID, BVid)$. This action is enabled if the status is $VERIFIED$. There are two possible predecessor actions that set the status to $VERIFIED$. The first is **SendTouchStone**$(eid, ts, BVid, VID)$, which retrieves a touchstone ts either from another broker or vendor, and this touchstone is used in the condition $if(verifyPW(ts, Pwords))$. This means that all paywords in the payword chain $Pwords$ and the received touchstone, belong to the same payword chain. If ts was obtained from a broker, the invariant holds trivially, if it was obtained from another vendor it holds because by assumption the old touchstone was a successor of a broker touchstone in a payword chain.

The other possible predecessor is input action **SendPaywords**$(cid, (eid, pwords, tloc), VID)$, which will check the condition $if\big(verifyPW(VendorDB(eid), pwords)\big)$ if the touchstone is in the vendors own database. This means that we can use the assumption that the old touchstone was a successor of a broker touchstone to show that the new touchstone is so as well.

Theorem 1: Let $(eid, pwords, vid) \in Ewallet$ of a customer $Customer(CID)$, then there exists a broker $Broker(BID)$ and an $n \in \mathbb{N}$ such that for the broker database holds $h^n(pwords_1) = BrokerDB(eid)$, where $pwords_1$ is the first element of the payword chain.

Proof: This follows from Lemma 1 to 3.

B. Non-Blocking Behavior

The automata that model of the Netpay protocol only use input and output actions, and no internal hidden action. For every action two automata have to synchronize. An action can take place if the precondition of the input and output action are satisfied. If the precondition of the output action is satisfied, but the precondition of the corresponding input action is not, then the output action is blocked. This means that one automaton can prevent another automaton from producing output.

In the I/O automata framework it is required that all input actions are always enabled, i.e. output action can never be blocked. In the interface automata framework states in which actions are blocked are identified and removed from the model. Neither of these two solution seemed appropriate for this case; output may be blocked, if for example the other automaton is processing another ecoin. This is acceptable as long as this output action is not blocked indefinitely. If the output action is enabled then at some point in the future the corresponding input action should be enabled as well. In the context of Netpay this means for example that a broker might block a customer for some time, while it processes a different ecoin request, but that the broker will eventually return to status $IDLE$, where it accepts the request.

To prove the non-blocking behavior we have to assume that the Netpay automaton model is (weak) fair, namely that if an action cannot be enabled indefinitely, i.e. the precondition of both input and output action are satisfied, then the action will eventually be taken. Given that assumption we can show that if an output action is indefinitely enabled, then the corresponding input action will be enabled eventually.

Key to this is the observation that all automata will return to the status $IDLE$. Most communication between two automata are models as a pair of actions. These pairs are:

- Shared actions **Send**(cid, n, BID) and **SendEcoin**$((Eid, Pwords, BID), Cid, BID)$; the request for an ecoin by a customer, and the reply by a

broker.

- Shared actions **GetTouchStone**$(eid, BVid, VID)$ and **GetTouchStone**$(Eid, BVid, VID)$; the request on a touchstone from broker/vendor and the reply.
- Shared actions **SendPaywords**$(CID, newcoin, vid)$ and **SendInformationBought**$(CID, BVid)$; the sending of an ecoin to a vendor, and the confirmation.

For the first two pairs we have that after taking the first action, the only action that both participating automata can take is the second, and the second action is enabled. This means it won't be blocked and the automata can return to $IDLE$. The third pair is slightly more involved. If the touchstone is held by the vendor, then the only action that is possible next is **SendInformationBought**$(CID, BVid)$, and that action is enabled. If the touchstone is with another vendor or broker, it has to execute action **GetTouchStone**$(eid, BVid, VID)$ and **GetTouchStone**$(Eid, BVid, VID)$. Since these are not blocking, the requesting vendor will reach status $VERIFIED$ eventually, which means that **SendInformationBought**$(CID, BVid)$ is enabled. This shows that for either of these series of actions, the automata will get back to a status that accepts new requests.

V. Conclusion and Future Research

In this paper, we have modeled and analyzed some properties of Netpay protocol using IA. We have verified two important properties:

- There is no deadlock in the model.
- If the protocol is executing according to its specification, the ecoins remains valid.

The verification done in this paper for ecoin validity is on the assumption that the customer and vendor is cooperative. We are currently working on verifying whether the ecoins remain valid when a customer or vendor is not cooperative. As mentioned in the paper, there are different e-wallets for the Netpay protocol, we will model and verify Netpay protocol with different e-wallets. The work of this paper has not addressed anonymity and robustness properties. These are two important properties of the Netpay protocol which is part of our future work.

References

[1] X. Dai and J. Grundy. Architecture for a component-based, plug-in micro-payment system. In In Proceedings of the Fifth Asia Pacific Web Conference. LNCS, Springer, April.

[2] X. Dai and J. Grundy. Three kinds of e-wallets for a netpay micro-payment system. In The Fifth International Conference on Web Information Systems Engineering, pages 66–67, Brisbane, Australia, November 22-24 2004. LNCS 3306.

[3] X. Dai and J. Grundy. Architecture of a micro-payment system for thin-client web applications. In In Proceedings of the 2002 International Conference on Internet Computing. CSREA Press, June 24-27.

[4] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The millicent protocol for inexpensive electronic commerce. In Fourth International World Wide Web Conference, December 1995.

[5] R. Rivest and A. Shamir. Payword and micromint: Two simple micropayment schemes. pages 307–314. LNCS, 1998.

[6] A. Herzberg and H. Yochai. Mini-pay: Charging per click on the web. 1996.

[7] R. Hauser, M. Steiner, and M. Waidner. Micro-payments based on ikp. In Proceedings of 14th Worldwide Congress on Computer and Communications Security Protection, pages 67– 82, Paris-La Defense, France, December 22-24 1996. Lecture Notes in Computer Science.

[8] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over the internet. the popcorn project. In 18th International Conference on Distributed Computing Systems (18th ICDCS'98), pages 592–601, Amsterdam, The Netherlands, 1998. IEEE.

[9] N. Lynch and M. Tuttle. An Introduction to Input/Output automata. Technical Memo MIT/LCS/TM-373, Massachusetts Institute of Technology, November 1988.

[10] X. Dai, J. Grundy, and B. Lo. Comparing and contrasting micro-payment models for e-commerce systems. In Proceedings of the International Conferences of Info-tech and Info-net (ICII), China, 2001.

[11] Y. Kawabe, K. Mano, H. Sakurada, and Y. Tsukada. Theorem proving anonymity of infinite systems. Information Processing Letters, 101(1):46–51, 2007.