An operational model of QuickPay

Pieter H. Hartel, Jake Hill and Matt Sims
phh@ecs.soton.ac.uk, {Jake.Hill, Matt.Sims}@bt-sys.bt.co.uk

Declarative Systems and Software Engineering Group
Technical Report DSSE-TR-98-4
June 12 1998

Department of Electronics and Computer Science
University of Southampton
Highfield, Southampton SO17 1BJ, United Kingdom

An operational model of QuickPay

Pieter H. Hartel * Jake Hill † Matt Sims †

June 12, 1998

Abstract

QuickPay is a system for micro payments aiming to avoid the cost of cryptographic operations during payments. An operational model of the system has been built to assist in the search for weaknesses in the protocols. As a result of this model building activity, one minor weakness has been found. Another more serious weakness has been rediscovered and a number of solutions are proposed. The strongest solution is proved correct.

1 Introduction

Electronic payment systems [7], like conventional payments systems, are designed to conserve money. However, if the economic value of a transaction is low, it may be acceptable on occasions to gain or loose payments. In a micro payment system, each transaction represents so little real value that the cost of preventing all gain or loss may not offset the advantages. We study a number of optimisations to an electronic payment system, ranging from a provably correct, lossless optimisation to efficient but lossy optimisations.

Most micro payment systems involve three parties: brokers, merchants and customers [2, 5, 8]. The role of the broker is to exchange 'real' money for tokens. The broker provides services to the customers and the merchants and as such should be the most trusted party of the system. The merchant delivers services or goods to the customer in exchange for tokens and as such should be trusted by the customers. Merchants do not need to be trusted by the brokers. The brokers and merchants provide services to the customers, which means that the customers must have trust in both, but no one has to trust a customer. The hierarchy in the trust model should be reflected in the protocols of the payment system.

QuickPay is a micro payment scheme with prepayment [1, 6]. The customer must register with the broker to obtain an electronic *carnet* with *value* tokens before a sale may take place. It is possible to acquire additional value tokens, or to refresh the value tokens at any time. A merchant must also register with the broker to obtain an electronic *till* with *authentication* tokens. This will enable the merchant to validate the tokens presented by the customer. Once customer and merchant are registered, a sale may proceed as follows. First the customer presents one or more value tokens to the merchant. The merchant then authenticates himself [1] with the broker and presents the customers value tokens to the broker for validation. The broker decides whether the customers tokens are valid. If the merchant is satisfied that the tokens are valid, goods/services may be delivered. The customer takes delivery but does not receive a receipt.

The QuickPay philosophy is to make transactions cheap in two ways. Firstly, offline payments are allowed although they are less secure than online payments. Secondly, payment does not require cryptographic computations. This is achieved by using real (as opposed to pseudo) random numbers as tokens. A sequence of random numbers is generated, encrypted and transferred when the customer or the merchant register with the broker. Creating the random numbers is efficient, as a hardware device (a noisy diode) is used rather than an computationally intensive algorithm. Encrypting and transmitting a sequence of random numbers can be relatively inefficient. However, this is only done during registration; during payment transactions only a single random number is transferred in clear. Systems that use cryptographic computations during every transaction, such as Millicent [2] and Mini-Pay [5], are inherently less efficient than QuickPay but possibly more secure.

Micro payment systems raise a number of interesting questions because they differ from normal payment systems. The present paper makes the following contributions to the understanding of micro payment systems in general, and to that of QuickPay in particular:

- to introduce an operational model for the QuickPay protocols, which allows real life scenarios to be studied at an appropriate level of abstraction.
- to investigate the correctness of an essential optimi-

*Dept. of Electronics and Computer Science, Univ. of Southampton, Email: phh@ecs.soton.ac.uk

†BT Laboratories, Martlesham Heath, Ipswich, UK, Email: {Jake.Hill, Matt.Sims}@bt-sys.bt.co.uk

[1] We refer to a customer using the words 'her' or 'she' and we refer to a merchant using the words 'his' or 'he'.

sation to the basic protocol, which allows multiple token payments to be replaced by a single token payment.

Many aspects of the real system are not modelled and consequently cannot be studied using the present model. The prime example is performance, which could be modelled but only by a more detailed description of the system than we present here.

The operational model of QuickPay is written with the aid of the `latos` [4] tool, which provides type checking and animation of specifications. The type checking facility of the tool has been used to avoid inconsistencies in the model, and the animation facilities have been used to explore various transaction sequences.

Many approaches have been used to analyse cryptographic protocols [10, pages 65–68]. Our work falls in the category of approaches that bring an existing formalism to bear on the specific problems of analysing cryptographic protocols. The criticism levied at this approach is that the formalisms are too general purpose and thus not specifically suitable to the task. We believe that this is not the case, and find support for our thesis in reports from a recent workshop. Ryan et al [9] survey the state of the art in model checking of security protocols, mainly using CSP but also Action systems and the B-method. The survey shows that specialised logics such as BAN logic are not necessary, other formal methods can be effective as well.

Another report from the same workshop by Gunter et al [3] reminds us that it is difficult to model a system in an appropriate mathematical framework, and the properties that one can prove apply to the model and not necessarily to the system. This caveat applies to our work also. In particular our correctness proof applies to the model and not necessarily to the QuickPay prototype. However, it would be a pleasant (but unlikely) surprise if the problems that we identified in the model do not also apply to the prototype.

The next section briefly presents the prototype implementation of QuickPay. Section 3 defines the model of QuickPay. Section 4 presents a number of case studies that show how QuickPay transactions are processed, and how incorrect use of the system can be prevented. Two problems and a number of solutions to one of the problems are explored in Section 5.

The last section presents conclusions and discusses further work.

2 Prototype

The QuickPay prototype has been tested on making payments over the internet. The prototype works as follows. The customer first starts her carnet application, which puts up a window to inform the customer about her balance as shown below.

The customer also starts up an unrelated application that might require payment, such as an intelligent agent that is going to make some purchases on her behalf. For the purpose of this example we will use a web browser, which is being used to select a page of information from a web server, for example `http://www.merchant.com`. Figure 1 shows the information provided by the server, as it appears on the customers work station. The Web page she is looking at is actually a schematic diagram of the QuickPay prototype. The diagram shows the three main parties and the (TCP/IP) connections between them.

If a page contains links that require payment, the web browser sends an http request to `http://till.merchant.com/`. The merchants till application opens a TCP/IP connection to the carnet at the customer site (identified by her IP address) and the carnet puts up a window asking the customer to confirm the sale. When the customer agrees, the merchant receives the tokens over the TCP/IP connection and clears them with the vault application at the brokers site. The carnet can be customised to designate merchants as permanently trusted, or trusted for the current session. Such merchants can help themselves to tokens without confirmation from the customer.

If the turn over of the merchant is high, a permanent TCP/IP connection (solid arrows) is used, otherwise a transient connection would be better. When the merchant is satisfied that the tokens have cleared, the till sends the page to the customer as a reply to the original http request.

The prototype implementation uses a UNIX platform for the vault application, the till application and the web server. The prototype offers Windows, Unix and MacOS implementations of the carnet.

The model to be described in the next section takes into account the core aspects of the implementation, but abstracts away from as much detail as possible. This strategy makes it possible to concentrate on the essentials, keeping the model simple and elegant. Once finished, it would be possible to refine the model so as to take on board more detail. Ultimately this process of refinement

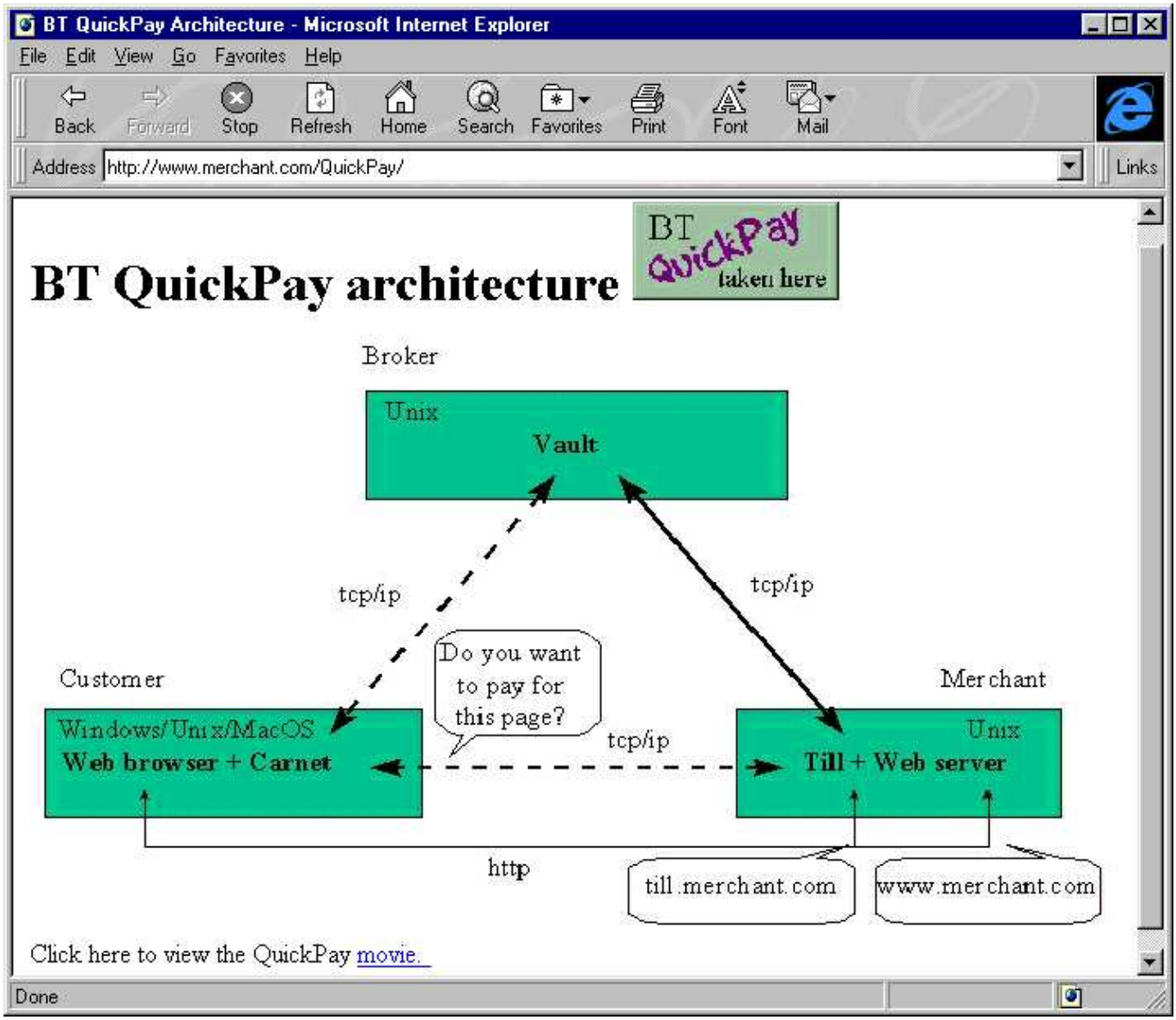

Figure 1: The architecture of the QuickPay prototype.

would lead to a finely detailed model that is able to describe every aspect of the implementation. The present work should be considered a starting point for the refinement process.

The model takes into account the transactions that rely on the TCP/IP connections, but abstracts away from the actual protocol implementation. The model also takes into account the three parties but it is not concerned with the Web browser/server. These components can be replaced by other client/server applications and are therefore not relevant to the model. The model abstracts away from the internal representations of data and messages in the carnet, till and vault applications.

3 Model

The model represents the QuickPay book keeping by a state, and the messages exchanged by the QuickPay protocols are represented by a list of transactions. Each transaction causes a transformation to be applied to the state, modelling the change in the book keeping as a result of a message exchange in the real system.

The state of the model is described by a number of data type definitions. The transactions that can take place are described by a set of logical inference rules operating on that data. Transactions and state are bound together in a configuration, which records the present state of the system as well as the sequence of transactions that have yet to take place. The collection of inference rules defines a relation over configurations. The model is animated by computing the transitive closure of the relation.

In subsequent sections we introduce the state (s), the transactions (tr), the relation ($\overset{qp}{\Rightarrow}$) and its closure ($\overset{qp}{\Rightarrow}*$).

3.1 State

The state s of the model is represented by a 3-tuple consisting of the book keeping of the customers (cs), brokers (bs), and merchants (ms).

$$\mathsf{s} \equiv (\mathsf{cs},\ \mathsf{bs},\ \mathsf{ms});$$

This rather innocent looking tuple represents the major abstraction of the model with respect to the prototype. The latter distributes the information with the protocol taking care that appropriate information is exchanged between the parties, but no more. The model in principles allows unlimited access to information. However, the model has been designed in such a way that it is easy to see (and prove) where and when information is accessed.

The basis of all book keeping is formed by numbers and names. These will be elaborated first. In subsequent sections, the state of each of the parties will be discussed in detail.

3.1.1 Scalar data

A payment system deals with numeric data and the naming of parties involved in transactions. The model uses n for counters; a for maintaining token account values and balances; r for random numbers; v for value tokens; and t for authentication tokens.

$$\begin{aligned}\mathsf{a} &\equiv \mathbb{N};\\ \mathsf{n} &\equiv \mathbb{N};\\ \mathsf{r} &\equiv \mathbb{N};\\ \mathsf{v} &\equiv \mathbb{N};\\ \mathsf{t} &\equiv \mathbb{N};\end{aligned}$$

The name of a customer is of type $\mathsf{id_c}$. The suffix $_{\mathsf{c}}$ indicates that this information pertains to customers only. Similarly the name of a merchant is of type $\mathsf{id_m}$ and the name of a broker is of type $\mathsf{id_b}$.

$$\begin{aligned}\mathsf{id_c} &\equiv \mathsf{string};\\ \mathsf{id_b} &\equiv \mathsf{string};\\ \mathsf{id_m} &\equiv \mathsf{string};\end{aligned}$$

3.1.2 Customer

The QuickPay prototype supports one broker doing business with any number of merchants and customers. This means that the customers and merchants do not need to know the identity of the broker. In the case of the customers' carnet c a representation consisting of just a list of tokens $[\mathsf{v}]$ is thus adequate. Customers are unique, so a representation of the customer data as a partial mapping cs from customer ids $\mathsf{id_c}$ to customer records c is appropriate.

$$\begin{aligned}\mathsf{c} &\equiv [\mathsf{v}];\\ \mathsf{cs} &\equiv \mathsf{id_c} \rightharpoonup \mathsf{c};\end{aligned}$$

3.1.3 Broker

The broker keeps a record of type b (below) containing three components. The first is a master list of random numbers $[\mathsf{r}]$. The second component is a partial mapping from customer ids to the records $([\mathsf{v}], \mathsf{a})$ kept by the broker on behalf of the customers. These records contain two components: a list of value tokens $[\mathsf{v}]$ and an account a. The third component of the broker record is a partial mapping from merchant ids to the records $([\mathsf{t}], \mathsf{a})$ kept by the broker on behalf of the merchants. These latter records also contain two components: a list of authentication tokens $[\mathsf{t}]$ and an account a.

$$\mathsf{b} \equiv ([\mathsf{r}],\ \mathsf{id_c} \rightharpoonup ([\mathsf{v}],\ \mathsf{a}),\ \mathsf{id_m} \rightharpoonup ([\mathsf{t}],\ \mathsf{a}));$$

Customer accounts function as a 'budget' and are decremented each time the customer purchases new tokens. Merchant accounts function as a 'balance' and are incremented each time the merchant clears some tokens.

This symmetry constitutes a slight deviation from the real QuickPay system, which only has merchant accounts.

We feel justified in making this deviation as the real system incorporates an interface to conventional payment systems, which will impose limits on the amount of tokens that customers can introduce. The budget is thought to model these external constraints, for it limits customer spending.

$$\mathsf{bs} \equiv \mathsf{id_b} \rightharpoonup \mathsf{b};$$

The type bs represents a partial mapping from broker ids $\mathsf{id_b}$ to broker records b, with the additional constraint that the domain of the mapping is a singleton set.

$$\#\mathsf{domain(bs)} = 1;$$

The constraint represents the fact the prototype has only one broker. Representing bs as a mapping is perhaps overkill, but we use it for symmetry reasons and to be able to explore scenarios where another broker infiltrates the system.

3.1.4 Merchant

The merchant has a till (or merchant record) m that contains a list of authentication tokens [t]. In addition, for each customer the merchant records a list of value tokens [v] received, but as yet uncleared by the broker.

The type ms represents a partial mapping from merchant ids $\mathsf{id_m}$ to merchant records m.

$$\mathsf{m} \equiv ([\mathsf{t}],\ \mathsf{id_c} \rightharpoonup [\mathsf{v}]);$$
$$\mathsf{ms} \equiv \mathsf{id_m} \rightharpoonup \mathsf{m};$$

This concludes the presentation of the state of the QuickPay model. The next section describes how the state is used in transactions.

3.2 Transactions

The model represents transactions as separate actions and reactions. This permits actions be modelled whilst the reaction is not forthcoming. Each transaction is labelled and carries a number of parameters to identify the parties involved. Some transactions require further information, such as a token count n. The definitions below represent 'control' information of the messages transmitted in the actual prototype; we abstract away from actual 'payload' of the real messages (i.e., the lists of tokens).

$$\begin{aligned}\mathsf{tr} \equiv\ & \mathbf{update}(\mathsf{id_c},\ \mathsf{id_b},\ \mathsf{n}) \mid \mathbf{sell}(\mathsf{id_m},\ \mathsf{id_c},\ \mathsf{n}) \mid \\ & \mathbf{clear}(\mathsf{id_b},\ \mathsf{id_m},\ \mathsf{id_c}) \mid \mathbf{stock}(\mathsf{id_m},\ \mathsf{id_b},\ \mathsf{n}) \mid \\ & \mathbf{auth_m}(\mathsf{id_m},\ \mathsf{id_b}) \mid \mathbf{auth_b}(\mathsf{id_b},\ \mathsf{id_m});\end{aligned}$$

Figure 2 illustrates which parties are involved in each of the transactions. Value tokens (indicated by thick arrows)

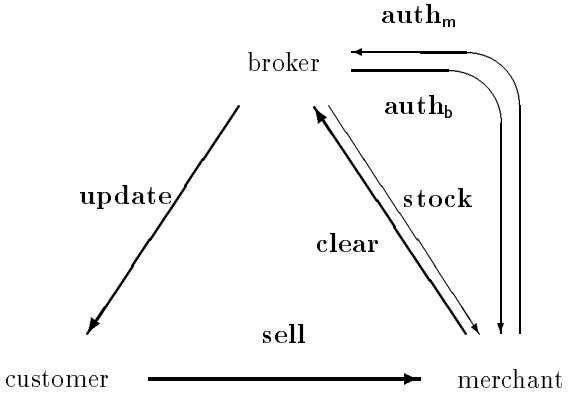

Figure 2: The QuickPay parties and transactions. Thick arrows represent value token transfer and thin arrows represent authentication token transfer.

flow from the broker to the customer (**update**) and then via the merchant (**sell**) back to the broker (**clear**). The authentication tokens (thin arrows) flow from the broker to the merchant (**stock** and $\mathbf{auth_b}$) and from the merchant to the broker ($\mathbf{auth_m}$).

The notion of a configuration augments the (static) state s with a (dynamic) list of transactions [tr]. We can now formulate the model of QuickPay as a relation $\overset{qp}{\Rightarrow}$ over configurations. The type of the relation is:

$$\overset{qp}{\Rightarrow} :: \langle[\mathsf{tr}],\ \mathsf{s}\rangle \leftrightarrow \langle[\mathsf{tr}],\ \mathsf{s}\rangle;$$

The relation itself is defined using a set of inference rules based on the following general pattern:

$$[\ldots]\ \frac{\vdash \ldots}{\vdash \langle \mathsf{tr} : \mathsf{trs},\ (\mathsf{cs},\ \mathsf{bs},\ \mathsf{ms})\rangle \overset{qp}{\Rightarrow} \langle \mathsf{trs},\ (\mathsf{cs}',\ \mathsf{bs}',\ \mathsf{ms}')\rangle},\ \mathbf{if}\ \ldots;$$

The conclusion of the rule (below the horizontal line) isolates the current transaction tr from the remaining transactions trs, and exposes the three components cs, bs and ms of the current state. The premises of the rule then assert a relationship between the current components of the state and the new components of the state (cs′, bs′ ms′). A rule applies only if the side condition **if** ... yields true.

The following sections present the detailed rules for each transaction.

3.2.1 Update transaction

The update transaction supplies the customer with value tokens. The customer is supposed to prepay these tokens using real money, a credit card or some other means.

In an implementation of QuickPay this may involve additional transactions, encryption of the tokens whilst in transit etc. These details are not considered relevant for the purpose of searching for weaknesses in the protocol and are thus not modelled here. However, the eventuality that tokens cannot be obtained is modelled using the customers budget.

In the transaction as specified by the rule $\mathsf{update}^{\mathsf{ok}}$ (below) the identities of the customer $\mathsf{id_c}$ and broker $\mathsf{id_b}$, as well as the number of tokens required n is given by the parameters of the **update** $(\mathsf{id_c}, \mathsf{id_b}, \mathsf{n})$ transaction.

The broker keeps a record for the customer, which is found by looking up the broker $\mathsf{id_b}$ in the mapping bs, and then within the mapping $\mathsf{cs_b}$ of the broker looking up the customer $\mathsf{id_c}$. This yields the customer data $(\mathsf{vs_c^b}, \mathsf{a_c})$ as viewed by the broker.

All mappings used in the model are partial. This means that for example $\mathsf{cs_b}(\mathsf{id_c})$ may be undefined, which would be the case when broker $\mathsf{id_b}$ and customer $\mathsf{id_c}$ do not know each other. This eventuality is covered by the conjunct $\mathsf{id_c} \in \mathsf{domain}(\mathsf{cs_b})$ of the side condition, which, together with the other conjuncts ensures that the lookup operations yield a well defined result. The last conjunct of the side condition $\mathsf{n} \leq \mathsf{a_c}$ asserts that the customer must not be over budget for the transaction to succeed. Since n represents a natural number, it is unnecessary to check for negative values.

There are two views on the customers carnet: $\mathsf{vs_c^b}$ is the brokers view on the customers list of tokens and $\mathsf{vs_c^c}$ would be the customers view on her own list of tokens. In general the superscripts indicate the view of a particular party on some information. The subscript indicates what the information represents. The two views on the customers carnet may differ, if the customer has spent tokens with a merchant who has not (yet) cleared the tokens with the broker. This creates the potential for misuse of the system, as illustrated by the double spending scenario given in Section 4.2.3.

The QuickPay protocol has been designed to be customer friendly in that the customer is allowed to refresh her carnet at any time. The idea is that a customer may be using different computers at different times. When she wishes to make a purchase, all she has to do is refresh her carnet. There is no need to carry the carnet around on some media. Even if she looses the computer with her carnet, she can simply refresh the carnet using another computer. The refresh automatically invalidates the lost tokens.

As a consequence, the model and the prototype implementation thus ignore the customers own view on the carnet. This is the reason why $\mathsf{vs_c^c}$ is not mentioned in the rule. Instead the update transaction creates a completely new carnet vs'. The carnet will contain the number of tokens $\#\mathsf{vs_c^b}$ kept in the current carnet (brokers view), plus the number of tokens n that is currently being (pre)paid for. The new number of tokens is represented by n'. The prime signifies the fact that this is new information. The new carnet contains n' random numbers. (The function take returns the first n' elements of the list rs. Similarly the function drop returns a list containing all elements of rs except the first n'.)

$$
\begin{array}{ll}
 & \vdash \mathsf{bs}(\mathsf{id_b}) = (\mathsf{rs},\ \mathsf{cs_b},\ \mathsf{ms_b}),\\
 & \vdash \mathsf{cs_b}(\mathsf{id_c}) = (\mathsf{vs_c^b},\ \mathsf{a_c}),\\
 & \vdash \#\mathsf{vs_c^b} + \mathsf{n} = \mathsf{n}',\\
 & \vdash \mathsf{take}(\mathsf{n}',\ \mathsf{rs}) = \mathsf{vs}',\\
 & \vdash \mathsf{vs}' = \mathsf{c}',\\
 & \vdash \mathsf{cs_b} \oplus \{\mathsf{id_c} \mapsto (\mathsf{vs}',\ \mathsf{a_c} - \mathsf{n})\} = \mathsf{cs_{b'}},\\
 & \vdash (\mathsf{drop}(\mathsf{n}',\ \mathsf{rs}),\ \mathsf{cs_{b'}},\ \mathsf{ms_b}) = \mathsf{b}'\\
\hline
[\mathsf{update}^{\mathsf{ok}}] & \vdash \langle \mathbf{update}(\mathsf{id_c},\ \mathsf{id_b},\ \mathsf{n}) : \mathsf{trs},\ (\mathsf{cs},\ \mathsf{bs},\ \mathsf{ms})\rangle \overset{qp}{\Rrightarrow}\\
 & \quad \langle \mathsf{trs},\ (\mathsf{cs} \oplus \{\mathsf{id_c} \mapsto \mathsf{c}'\},\ \mathsf{bs} \oplus \{\mathsf{id_b} \mapsto \mathsf{b}'\},\ \mathsf{ms})\rangle,\\
 & \mathbf{if}\ \mathsf{id_c} \in \mathsf{domain}(\mathsf{cs}) \wedge \mathsf{id_b} \in \mathsf{domain}(\mathsf{bs}) \wedge\\
 & \quad \mathsf{id_c} \in \mathsf{domain}(\mathsf{cs_b}) \wedge \mathsf{n} \leq \mathsf{a_c};
\end{array}
$$

$$[\mathsf{update}^{\mathsf{nok}}]\ \vdash \langle \mathbf{update}(\mathsf{id_c},\ \mathsf{id_b},\ \mathsf{n}) : \mathsf{trs},\ \mathsf{s}\rangle \overset{qp}{\Rrightarrow} \langle \mathsf{trs},\ \mathsf{s}\rangle;$$

The new customer record is c', the new mapping of customer ids to customer records is $\mathsf{cs_{b'}}$ and the new broker record is b'. (The operator $\oplus$ represents functional overriding). The state components representing customer and broker records are updated but the component representing merchants ms is unchanged. This shows that gathering the state of all parties in a single tuple, as discussed in Section 3.1, is safe.

The model differs from the prototype in that the latter has been optimised to re-use random numbers, by hashing each number with a key specific to the customer. It was felt appropriate for the level of abstraction required to ignore this optimisation and use a fresh batch of random numbers each time. Both the model and the prototype assume all random numbers to be different, and that there is a sufficient supply of random numbers.

In a system with more than one broker, the above transaction would be unsafe. Suppose that a customer has accounts with two brokers, A and B. She could then refresh the carnet obtained from A with B and vice versa.

3.2.2 Sell transaction

The sell transaction initiates a sale of goods from the merchant to the customer. For a sale, the merchant $\mathsf{id_m}$ looks up his record for customer $\mathsf{id_c}$. This record $\mathsf{vs_c^m}$ consists of a list of value tokens, which is appended to the required number of tokens n (using the $+\!\!+$ operator). The customer is relieved of the same number of tokens.

$$
\frac{\begin{array}{l}
\vdash \mathsf{ms}(\mathsf{id_m}) = (\mathsf{ts_m},\ \mathsf{cs_m}),\\
\vdash \mathsf{cs}(\mathsf{id_c}) = \mathsf{vs_c^c},\\
\vdash \mathsf{cs_m}(\mathsf{id_c}) = \mathsf{vs_c^m},\\
\vdash \mathsf{drop}(\mathsf{n},\ \mathsf{vs_c^c}) = \mathsf{c}',\\
\vdash \mathsf{cs_m} \oplus \{\mathsf{id_c} \mapsto (\mathsf{take}(\mathsf{n},\ \mathsf{vs_c^c}) +\!\!+ \mathsf{vs_c^m})\} = \mathsf{cs_{m'}},\\
\vdash (\mathsf{ts_m},\ \mathsf{cs_{m'}}) = \mathsf{m}'
\end{array}}{\begin{array}{ll}
[\mathsf{sell^{ok}}] & \vdash \langle \mathbf{sell}(\mathsf{id_m},\ \mathsf{id_c},\ \mathsf{n}) : \mathsf{trs},\ (\mathsf{cs},\ \mathsf{bs},\ \mathsf{ms})\rangle \stackrel{qp}{\Rightarrow}\\
& \quad \langle \mathsf{trs},\ (\mathsf{cs} \oplus \{\mathsf{id_c} \mapsto \mathsf{c}'\},\ \mathsf{bs},\ \mathsf{ms} \oplus \{\mathsf{id_m} \mapsto \mathsf{m}'\})\rangle,\\
& \quad \mathbf{if}\ \mathsf{id_m} \in \mathsf{domain}(\mathsf{ms}) \wedge \mathsf{id_c} \in \mathsf{domain}(\mathsf{cs}) \wedge\\
& \qquad \mathsf{id_c} \in \mathsf{domain}(\mathsf{cs_m}) \wedge \mathsf{n} \leq \#\mathsf{vs_c^c};
\end{array}}
$$

$$
[\mathsf{sell^{nok}}]\ \vdash \langle \mathbf{sell}(\mathsf{id_m},\ \mathsf{id_c},\ \mathsf{n}) : \mathsf{trs},\ \mathsf{s}\rangle \stackrel{qp}{\Rightarrow} \langle \mathsf{trs},\ \mathsf{s}\rangle;
$$

The sell transaction fails if either the customer and the merchant do not know each other, or if the customers carnet $\mathsf{vs_c^c}$ does not at least contain the required number n of tokens. To validate the tokens, a number of further steps are required (**auth**$_\mathsf{b}$, **auth**$_\mathsf{m}$ and **clear**, see below).

3.2.3 Stock transaction

Stock is for the merchant what update is for the customer. The stock transaction replenishes the merchants till with a fresh list of authentication tokens $\mathsf{ts_{m'}}$. The tokens are used to authenticate the merchant to the broker, and vice versa.

The old list of tokens is not needed, which is indicated by the wild-card _. The merchant does not pay for these tokens via conventional means, and is thus free to stock new tokens as often as he likes. The count n therefore gives the absolute number of fresh tokens, as opposed to the relative number for the customers **update** transaction.

$$
\frac{\begin{array}{l}
\vdash \mathsf{bs}(\mathsf{id_b}) = (\mathsf{rs},\ \mathsf{cs_b},\ \mathsf{ms_b}),\\
\vdash \mathsf{ms}(\mathsf{id_m}) = (\mathsf{ts_m^m},\ \mathsf{cs_m}),\\
\vdash \mathsf{ms_b}(\mathsf{id_m}) = (\_,\ \mathsf{a_m}),\\
\vdash \mathsf{take}(\mathsf{n},\ \mathsf{rs}) = \mathsf{ts_{m'}},\\
\vdash (\mathsf{ts_{m'}},\ \mathsf{cs_m}) = \mathsf{m}',\\
\vdash \mathsf{ms_b} \oplus \{\mathsf{id_m} \mapsto (\mathsf{ts_{m'}},\ \mathsf{a_m})\} = \mathsf{ms_{b'}},\\
\vdash (\mathsf{drop}(\mathsf{n},\ \mathsf{rs}),\ \mathsf{cs_b},\ \mathsf{ms_{b'}}) = \mathsf{b}'
\end{array}}{\begin{array}{ll}
[\mathsf{stock^{ok}}] & \vdash \langle \mathbf{stock}(\mathsf{id_m},\ \mathsf{id_b},\ \mathsf{n}) : \mathsf{trs},\ (\mathsf{cs},\ \mathsf{bs},\ \mathsf{ms})\rangle \stackrel{qp}{\Rightarrow}\\
& \quad \langle \mathsf{trs},\ (\mathsf{cs},\ \mathsf{bs} \oplus \{\mathsf{id_b} \mapsto \mathsf{b}'\},\ \mathsf{ms} \oplus \{\mathsf{id_m} \mapsto \mathsf{m}'\})\rangle,\\
& \quad \mathbf{if}\ \mathsf{id_b} \in \mathsf{domain}(\mathsf{bs}) \wedge \mathsf{id_m} \in \mathsf{domain}(\mathsf{ms}) \wedge\\
& \qquad \mathsf{id_m} \in \mathsf{domain}(\mathsf{ms_b});
\end{array}}
$$

$$
[\mathsf{stock^{nok}}]\ \vdash \langle \mathbf{stock}(\mathsf{id_m},\ \mathsf{id_b},\ \mathsf{n}) : \mathsf{trs},\ \mathsf{s}\rangle \stackrel{qp}{\Rightarrow} \langle \mathsf{trs},\ \mathsf{s}\rangle;
$$

The stock transaction can only fail when the merchant and the broker do not know each other.

3.2.4 Merchant authenticate transaction

Prior to clearing customer tokens, the merchant must authenticate himself with the broker. This is done by comparing the next authentication token from the merchant $\mathsf{hd}(\mathsf{ts_m^m})$ with the next token from the broker $\mathsf{hd}(\mathsf{ts_m^b})$. (The function hd returns the head element of a list; the function tl returns a list without its head element; an empty list is indicated by [].) If the merchant and the broker hold at least one authentication token each and if those tokens are the same, the merchant is deemed genuine. The clause $\mathsf{auth_m^{nok}}$ applies if the merchant is fraudulent, or if the merchant has run out of authentication tokens, or if the broker and the merchant do not know each other.

$$
\frac{\begin{array}{l}
\vdash \mathsf{bs}(\mathsf{id_b}) = (\mathsf{rs},\ \mathsf{cs_b},\ \mathsf{ms_b}),\\
\vdash \mathsf{ms}(\mathsf{id_m}) = (\mathsf{ts_m^m},\ \mathsf{cs_m}),\\
\vdash \mathsf{ms_b}(\mathsf{id_m}) = (\mathsf{ts_m^b},\ \mathsf{a_b}),\\
\vdash (\mathsf{tl}(\mathsf{ts_m^m}),\ \mathsf{cs_m}) = \mathsf{m}',\\
\vdash \mathsf{ms_b} \oplus \{\mathsf{id_m} \mapsto (\mathsf{tl}(\mathsf{ts_m^b}),\ \mathsf{a_b})\} = \mathsf{ms_{b'}},\\
\vdash (\mathsf{rs},\ \mathsf{cs_b},\ \mathsf{ms_{b'}}) = \mathsf{b}'
\end{array}}{\begin{array}{ll}
[\mathsf{auth_m^{ok}}] & \vdash \langle \mathbf{auth}_\mathsf{m}(\mathsf{id_m},\ \mathsf{id_b}) : \mathsf{trs},\ (\mathsf{cs},\ \mathsf{bs},\ \mathsf{ms})\rangle \stackrel{qp}{\Rightarrow}\\
& \quad \langle \mathsf{trs},\ (\mathsf{cs},\ \mathsf{bs} \oplus \{\mathsf{id_b} \mapsto \mathsf{b}'\},\ \mathsf{ms} \oplus \{\mathsf{id_m} \mapsto \mathsf{m}'\})\rangle,\\
& \quad \mathbf{if}\ \mathsf{id_b} \in \mathsf{domain}(\mathsf{bs}) \wedge \mathsf{id_m} \in \mathsf{domain}(\mathsf{ms}) \wedge\\
& \qquad \mathsf{id_m} \in \mathsf{domain}(\mathsf{ms_b}) \wedge\\
& \qquad \mathsf{ts_m^b} \neq [] \wedge \mathsf{ts_m^m} \neq [] \wedge \mathsf{hd}(\mathsf{ts_m^b}) = \mathsf{hd}(\mathsf{ts_m^m});
\end{array}}
$$

$$
[\mathsf{auth_m^{nok}}]\ \vdash \langle \mathbf{auth}_\mathsf{m}(\mathsf{id_m},\ \mathsf{id_b}) : \mathsf{trs},\ \mathsf{s}\rangle \stackrel{qp}{\Rightarrow} \langle \mathsf{trs},\ \mathsf{s}\rangle;
$$

The specification above assumes that when authentication fails, the first tokens of $\mathsf{ts_m^b}$ and $\mathsf{ts_m^m}$ are not considered 'spent'. This may or may not be the behaviour of the prototype; the point is that appropriate behaviour can be specified.

3.2.5 Broker authenticate transaction

The authenticate broker transaction is almost the same as the authenticate merchant transaction. The only difference between **auth**$_\mathsf{b}$ and **auth**$_\mathsf{m}$ is that the parameters are swapped. It may seem surprising that the two transactions are so similar, but this is a direct consequence of the asymmetric design of the protocol, which uses the same sequence of random numbers for both authentication steps.

$$
\frac{\vdash \langle \mathbf{auth}_\mathsf{m}(\mathsf{id_m},\ \mathsf{id_b}) : \mathsf{trs},\ \mathsf{s}\rangle \stackrel{qp}{\Rightarrow} \langle \mathsf{trs}',\ \mathsf{s}'\rangle}{[\mathsf{auth_b}]\ \vdash \langle \mathbf{auth}_\mathsf{b}(\mathsf{id_b},\ \mathsf{id_m}) : \mathsf{trs},\ \mathsf{s}\rangle \stackrel{qp}{\Rightarrow} \langle \mathsf{trs}',\ \mathsf{s}'\rangle;}
$$

3.2.6 Clear transaction

The clear transaction allows the merchant to clear the tokens from a given customer with the broker. This transaction is supposed to take place only when the merchant and the broker have just authenticated each other.

The side condition checks that the value tokens are indeed amongst those $\mathsf{vs_c^b}$ originally handed out to the customer.

$$
\begin{array}{l}
\vdash \mathsf{bs}(\mathsf{id_b}) = (\mathsf{rs},\ \mathsf{cs_b},\ \mathsf{ms_b}),\\
\vdash \mathsf{ms}(\mathsf{id_m}) = (\mathsf{ts_m^m},\ \mathsf{cs_m}),\\
\vdash \mathsf{ms_b}(\mathsf{id_m}) = (\mathsf{ts_m^b},\ \mathsf{a_m}),\\
\vdash \mathsf{cs_m}(\mathsf{id_c}) = \mathsf{vs_c^m},\\
\vdash \mathsf{cs_b}(\mathsf{id_c}) = (\mathsf{vs_c^b},\ \mathsf{a_c}),\\
\vdash \mathsf{vs_c^m} = \mathsf{vs_c^{m'}},\\
\vdash (\mathsf{ts_m^m},\ \mathsf{cs_m} \oplus \{\mathsf{id_c} \mapsto []\}) = \mathsf{m'},\\
\vdash \mathsf{vs_c^b} \backslash \mathsf{vs_c^{m'}} = \mathsf{vs_c^{b'}},\\
\vdash \mathsf{ms_b} \oplus \{\mathsf{id_m} \mapsto (\mathsf{ts_m^b},\ \mathsf{a_m} + \#\mathsf{vs_c^{m'}})\} = \mathsf{ms_{b'}},\\
\vdash \mathsf{cs_b} \oplus \{\mathsf{id_c} \mapsto (\mathsf{vs_c^{b'}},\ \mathsf{a_c})\} = \mathsf{cs_{b'}},\\
\vdash (\mathsf{rs},\ \mathsf{cs_{b'}},\ \mathsf{ms_{b'}}) = \mathsf{b'}\\
\hline
[\mathsf{clear^{ok}}]\ \vdash \langle \mathbf{clear}(\mathsf{id_b},\ \mathsf{id_m},\ \mathsf{id_c}) : \mathsf{trs},\ (\mathsf{cs},\ \mathsf{bs},\ \mathsf{ms})\rangle \overset{qp}{\Rightarrow}\\
\quad \langle \mathsf{trs},\ (\mathsf{cs},\ \mathsf{bs} \oplus \{\mathsf{id_b} \mapsto \mathsf{b'}\},\ \mathsf{ms} \oplus \{\mathsf{id_m} \mapsto \mathsf{m'}\})\rangle,\\
\quad \mathbf{if}\ \mathsf{id_b} \in \mathsf{domain}(\mathsf{bs}) \wedge \mathsf{id_m} \in \mathsf{domain}(\mathsf{ms}) \wedge\\
\quad\quad \mathsf{id_m} \in \mathsf{domain}(\mathsf{ms_b}) \wedge \mathsf{id_c} \in \mathsf{domain}(\mathsf{cs_m}) \wedge\\
\quad\quad \mathsf{id_c} \in \mathsf{domain}(\mathsf{cs_b}) \wedge \mathsf{vs_c^{m'}} \subseteq \mathsf{vs_c^b};
\end{array}
$$

$$
[\mathsf{clear^{nok}}]\ \vdash \langle \mathbf{clear}(\mathsf{id_b},\ \mathsf{id_m},\ \mathsf{id_c}) : \mathsf{trs},\ \mathsf{s}\rangle \overset{qp}{\Rightarrow} \langle \mathsf{trs},\ \mathsf{s}\rangle;
$$

The premise $\vdash\ \mathsf{vs_c^m} = \mathsf{vs_c^{m'}}$ does not appear to be very useful. It has been introduced only to facilitate the presentation of the optimisation in Section 5.3.

The clause $\mathsf{clear^{nok}}$ applies when the merchant tries to clear false tokens, or if one of the parties is unknown to the others.

3.3 Closure

The collection of inference rules presented above define a relation $\overset{qp}{\Rightarrow}$ over configurations.

The model is animated by computing the transitive closure $\overset{qp}{\Rightarrow}{}^*$ of the relation as shown by the function animate below. This function takes an initial configuration and delivers a list of configurations showing the remaining transactions and the state of the system after each transaction. The initial configuration is prepended to the result to show the starting point of the animation.

$$
\begin{array}{l}
\mathsf{animate} :: \langle [\mathsf{tr}],\ \mathsf{s}\rangle \rightarrow [\langle [\mathsf{tr}],\ \mathsf{s}\rangle];\\
\mathsf{animate}\langle \mathsf{trs},\ \mathsf{s}\rangle = \langle \mathsf{trs},\ \mathsf{s}\rangle : \langle \mathsf{trs},\ \mathsf{s}\rangle \overset{qp}{\Rightarrow} *;
\end{array}
$$

A complete model of QuickPay has now been established. The next section presents some examples of use.

4 Case studies

To assess the validity of the model, we will explore the behaviour of the protocols by animating a number of scenarios.

4.1 Sample state

The case studies use three potential customers, a broker and two merchants. The initial state of the system is such that:

- All token lists are initially empty.
- The merchant and customer accounts as held by the broker are initialised.
- The master list of random numbers held by the broker is initialised.

The initial state $\mathsf{s_0}$ is:

$$
\begin{array}{l}
\mathsf{s_0} :: \mathsf{s};\\
\mathsf{s_0} = (\mathsf{cs_0},\ \mathsf{bs_0},\ \mathsf{ms_0});
\end{array}
$$

The three customers are alice, bob, and carol. Each initially has an empty carnet.

$$
\begin{array}{l}
\mathsf{cs_0} :: \mathsf{cs};\\
\mathsf{cs_0} = \{\langle \mathsf{alice} \mapsto ([])\rangle,\ \langle \mathsf{bob} \mapsto ([])\rangle,\ \langle \mathsf{carol} \mapsto ([])\rangle\};
\end{array}
$$

The broker bank1 knows all about the two merchants and the three customers. The customers each have a budget of 100, the initial accounts of the merchants are empty. In the animations of the model we have used subsequences of the natural numbers by way of random sequence. (The notation $[\mathsf{t_x} \mid \mathsf{x} \leftarrow \mathsf{s}]$ causes x to range over the elements of the sequence s and returns a new sequence with elements $\mathsf{t_x}$.)

$$
\begin{array}{l}
\mathsf{bs_0} :: \mathsf{bs};\\
\mathsf{bs_0} = \{\langle \mathsf{bank1} \mapsto \mathsf{b_1}\rangle\}\\
\quad \mathbf{where}\\
\quad \mathsf{b_1} = (\mathsf{rs},\ \mathsf{cs},\ \mathsf{ms})\\
\quad\quad \mathbf{where}\\
\quad\quad \mathsf{rs} = [\mathsf{r} \mid \mathsf{r} \leftarrow [10..19]];\\
\quad\quad \mathsf{cs} = \{\langle \mathsf{alice} \mapsto ([],\ 100)\rangle,\\
\quad\quad\quad \langle \mathsf{bob} \mapsto ([],\ 100)\rangle,\\
\quad\quad\quad \langle \mathsf{carol} \mapsto ([],\ 100)\rangle\};\\
\quad\quad \mathsf{ms} = \{\langle \mathsf{shopy} \mapsto ([],\ 0)\rangle,\\
\quad\quad\quad \langle \mathsf{shopx} \mapsto ([],\ 0)\rangle\};\\
\quad\quad ;\\
\quad ;
\end{array}
$$

There are two merchants shopx and shopy in the system. The first knows about all three customers, the second knows only about alice.

$$
\begin{array}{l}
\mathsf{ms_0} :: \mathsf{ms};\\
\mathsf{ms_0} = \{\langle \mathsf{shopx} \mapsto ([],\ \mathsf{cs_x})\rangle,\ \langle \mathsf{shopy} \mapsto ([],\ \mathsf{cs_y})\rangle\}\\
\quad \mathbf{where}\\
\quad \mathsf{cs_x} = \{\langle \mathsf{alice} \mapsto []\rangle,\ \langle \mathsf{bob} \mapsto []\rangle,\\
\quad\quad \langle \mathsf{carol} \mapsto []\rangle\};\\
\quad \mathsf{cs_y} = \{\langle \mathsf{alice} \mapsto []\rangle\};\\
\quad ;
\end{array}
$$

4.2 Scenarios

The scenarios to be discussed include a correct 'run' of the protocol, and a number of incorrect ones.

4.2.1 Scenario 1: successful sale

In the first example shopx receives the counter value of 2 tokens from alice. An animation of the sale consists of a series of snapshots, which shows in detail how each transaction alters the state of the system. To save space customers or merchants with empty token lists are suppressed. Similarly, if the book keeping of any customer, broker or merchant is unaffected by a transaction then that information is suppressed.
The first step initialises the lists of random numbers for the broker.

bank1 [10 11 12 13 14 15 16 17 18 19]

$\rightarrow \mathsf{update}(\mathsf{alice}, \mathsf{bank1}, 3) \rightarrow$
The customers carnet contains three value tokens. The brokers vault contains the duplicates and shows that the customers budget is now 97.

alice [10 11 12]
bank1 [13 14 15 16 17 18 19]
alice ([10 11 12],97)

$\rightarrow \mathsf{stock}(\mathsf{shopx}, \mathsf{bank1}, 4) \rightarrow$
The till of shopx contains four authentication tokens. The brokers vault contains the duplicates and shows that the merchants account is 0.

bank1 [17 18 19]
alice ([10 11 12],97)
shopx ([13 14 15 16],0)
shopx [13 14 15 16]

$\rightarrow \mathsf{sell}(\mathsf{shopx}, \mathsf{alice}, 2) \rightarrow$
The merchant has received the two tokens $[10, 11]$ from the customer.

alice [12]
shopx [13 14 15 16]
alice [10 11]

$\rightarrow \mathsf{auth}_\mathsf{m}(\mathsf{shopx}, \mathsf{bank1}) \rightarrow$
The broker and the merchant agree that 13 is the first authentication token.

bank1 [17 18 19]
alice ([10 11 12],97)
shopx ([14 15 16],0)
shopx [14 15 16]
alice [10 11]

$\rightarrow \mathsf{auth}_\mathsf{b}(\mathsf{bank1}, \mathsf{shopx}) \rightarrow$
The merchant and the broker agree that 14 is the second authentication token.

bank1 [17 18 19]
alice ([10 11 12],97)
shopx ([15 16],0)
shopx [15 16]
alice [10 11]

$\rightarrow \mathsf{clear}(\mathsf{bank1}, \mathsf{shopx}, \mathsf{alice}) \rightarrow$
The merchant account has been credited with 2 tokens.

bank1 [17 18 19]
alice ([12],97)
shopx ([15 16],2)
shopx [15 16]

4.2.2 Scenario 2: price too high

There are several possibilities for turning the previous scenario into an unsuccessful sale. The scenario below differs from the previous in that the price of the product is now 5 tokens instead of 2.

bank1 [10 11 12 13 14 15 16 17 18 19]

$\rightarrow \mathsf{update}(\mathsf{alice}, \mathsf{bank1}, 3) \rightarrow$

alice [10 11 12]
bank1 [13 14 15 16 17 18 19]
alice ([10 11 12],97)

$\rightarrow \mathsf{sell}(\mathsf{shopx}, \mathsf{alice}, 5) \rightarrow$
The $\mathbf{sell}^{\mathbf{nok}}$ rule now applies because the customer only has three tokens $[10, 11, 12]$.

4.2.3 Scenario 3: double spending

QuickPay has been designed to be customer friendly. She can refresh her carnet at any time, and even if she looses the carnet, she promptly receives fresh tokens. We will now use the model to study a scenario which mis-uses this facility. Suppose that a customer makes a purchase, and then immediately refreshes her carnet. Refreshing the carnet will invalidate the tokens just offered to the merchant, so that the latter is unable to clear the tokens. The customer will only benefit from her bad behaviour if the merchant delivers the goods/services before clearing the tokens. The merchant may wish to do so in order to clear tokens in batch, and thus to amortise the cost of a clear transaction over a larger number of tokens.
The first step initialises the list of random numbers for the broker (called bank1), as shown below:

bank1 [10 11 12 13 14 15 16 17 18 19]

$\rightarrow$ update(alice, bank1, 2) $\rightarrow$
After the update transaction, the carnet of customer alice contains two value tokens, [10, 11]. The brokers vault contains the duplicates and shows that the customers budget is now 98:

alice [10 11]
bank1 [12 13 14 15 16 17 18 19]
alice ([10 11],98)

$\rightarrow$ stock(shopx, bank1, 4) $\rightarrow$
After the stock transaction, the till of the merchant shopx contains four authentication tokens, [12, 13, 14, 15]. The brokers vault contains the duplicates and shows that the merchants account is 0:

bank1 [16 17 18 19]
alice ([10 11],98)
shopx ([12 13 14 15],0)
shopx [12 13 14 15]

$\rightarrow$ sell(shopx, alice, 1) $\rightarrow$
The merchant has received the token 10 as payment from the customer:

alice [11]
shopx [12 13 14 15]
alice [10]

$\rightarrow$ update(alice, bank1, 0) $\rightarrow$
The customer updates her carnet, receiving two fresh tokens [16, 17]. The customer's budget is still 98. The broker's record also shows the new state of the carnet.

alice [16 17]
bank1 [18 19]
alice ([16 17],98)
shopx ([12 13 14 15],0)

$\rightarrow$ sell(shopx, alice, 1) $\rightarrow$
The customer makes another purchase, this time spending token 16. The merchant holds the old, invalid token 10 as well as a new, valid token 16:

alice [17]
shopx [12 13 14 15]
alice [16 10]

$\rightarrow$ auth_m(shopx, bank1) $\rightarrow$
The broker accepts that the merchant is authentic by agreeing that 12 is the first authentication token:

bank1 [18 19]
alice ([16 17],98)
shopx ([13 14 15],0)
shopx [13 14 15]
alice [16 10]

$\rightarrow$ auth_b(bank1, shopx) $\rightarrow$
The merchant also accepts that the broker is authentic, using token 13:

bank1 [18 19]
alice ([16 17],98)
shopx ([14 15],0)
shopx [14 15]
alice [16 10]

$\rightarrow$ clear(bank1, shopx, alice) $\rightarrow$
At this stage the clearing fails because [16, 10] does not match [16, 17]. The merchant account is not credited.

4.2.4 Scenario 4: fail to authenticate

A merchant may fail to authenticate, for example if the customer and the merchant are not using the same broker and the merchant uses the customers broker instead of his own.

For the purpose of this example, we introduce a second broker bank2 into the system. The new broker knows only about shopx. Strictly speaking, this violates the constraint of Section 3.1.3, which states that there is only one broker in the system.

bank1 [10 11 12 13 14 15 16 17 18 19]
bank2 [20 21 22 23 24 25 26 27 28 29]

$\rightarrow$ update(alice, bank1, 2) $\rightarrow$

alice [10 11]
bank1 [12 13 14 15 16 17 18 19]
alice ([10 11],98)

$\rightarrow$ stock(shopx, bank2, 3) $\rightarrow$

bank2 [23 24 25 26 27 28 29]
shopx ([20 21 22],0)
shopx [20 21 22]

$\rightarrow$ sell(shopx, alice, 1) $\rightarrow$

alice [11]
shopx [20 21 22]
alice [10]

$\rightarrow$ auth_m(shopx, bank1) $\rightarrow$
The authentication fails because bank1 does not know shopx.

Having explored some examples of the behaviour of the system we are now ready to study two problematic behaviours in more detail.

5 Problems and solutions

One of the main concerns in the design of QuickPay has been to make the protocols as efficient as possible. This has resulted in a design that uses random numbers instead of compute intensive cryptography and small messages instead of large ones. The strive for efficiency has brought with it some (potential) problems that we should like to address in this section.

5.1 Asymmetric authentication

We discovered a hitherto unknown problem whilst building the model of QuickPay. The **stock** transaction serves to provide the merchant with authentication tokens. These tokens are used both to authenticate the broker with the merchant and vice versa. It is conceivable that a broker may take advantage of the knowledge how these tokens are computed to trick the merchant. This danger would not arise if both the broker and the merchant would create their own list of tokens, which they would then use to authenticate the other party. We have not found a practical example that exploits this weakness.

5.2 Selecting lossy compression

To reduce the amount of data transmitted during a multiple token transfer, the prototype implementation of QuickPay optimises payments of $n > 1$ value tokens in the following way. Instead of transferring a list of tokens $[v_1, \ldots v_n]$ the implementation transfers just the last one, v_n. We have termed this the *selecting lossy compression*, because the optimisation compresses by selecting a token.

When clearing, the broker has a duplicate of the customers carnet. The broker is thus able to look the token v_n up in the duplicate. Normally, the token will be found at the n-th place, thus confirming both that the token is valid, and that it represents n tokens.

The scenario below shows why the optimisation is not correct.

5.2.1 Scenario 5: business with more merchants

In the scenario below, alice first makes two purchases, one at shopx and the second at shopy. The latter then successfully clears the token he has has just received from alice. This is shown in the scenario below. However, with the selecting lossy compression enabled, the broker would find the token received from shopy at position 2 in the duplicate of alice's carnet. As a result shopy would be credited by two tokens instead of just one. If shopx then tries to clear its token, the broker will claim that it has already been cleared, and refuse shopx its dues.

bank1 [10 11 12 13 14 15 16 17 18 19]

→ update(alice, bank1, 3) →

alice [10 11 12]
bank1 [13 14 15 16 17 18 19]
alice ([10 11 12],97)

→ stock(shopx, bank1, 4) →

bank1 [17 18 19]
alice ([10 11 12],97)
shopx ([13 14 15 16],0)
shopx [13 14 15 16]

→ stock(shopy, bank1, 4) →

bank1 []
alice ([10 11 12],97)
shopx ([13 14 15 16],0)
shopy ([17 18 19],0)
shopy [17 18 19]

→ sell(shopx, alice, 1) →

alice [11 12]
shopx [13 14 15 16]
alice [10]

→ sell(shopy, alice, 1) →

alice [12]
shopy [17 18 19]
alice [11]

→ $\mathsf{auth_m}$(shopy, bank1) →

bank1 []
alice ([10 11 12],97)
shopx ([13 14 15 16],0)
shopy ([18 19],0)
shopy [18 19]
alice [11]

→ $\mathsf{auth_b}$(bank1, shopy) →

bank1 []
alice ([10 11 12],97)
shopx ([13 14 15 16],0)
shopy ([19],0)
shopy [19]
alice [11]

→ clear(bank1, shopy, alice) →
At this stage we see that the token 11 in the merchants records for alice occurs in the second place in the brokers duplicate of alice's carnet.

Whilst building the QuickPay prototype we (re) discovered that the selecting lossy compression optimisation was incorrect. We also found a different optimisation (adding

lossy compression) which causes the scenario above to behave correctly. To study this new optimisation we applied it to the model, uncovering another problem. This lead us to a generalisation of compressing tokens, as well as a range of optimisations.

This is the subject of the next section.

5.3 A correct solution

The idea of the new optimisation is to compress tokens in a lossless way by treating some function of the value tokens as a new compressed token (of type $\bar{\mathsf{v}}$ say):

$$\mathsf{compress} :: [\mathsf{v}] \rightarrow \bar{\mathsf{v}};$$

The optimisation is correct only if there exists an expand operation, which recovers the original tokens from the compressed token. Therefore, the specification of expand is:

$$\begin{array}{l} \mathsf{expand} \quad :: \bar{\mathsf{v}} \rightarrow [\mathsf{v}]; \\ \mathsf{expand.compress} = \mathsf{id}; \end{array}$$

To realise the lossless compression optimisation three changes have to be made to the model.

- The merchant now has to record the new tokens $\bar{\mathsf{v}}$. This changes the definitions of m, ms and s from Section 3.1 to:

$$\begin{array}{l} \bar{\mathsf{m}} \equiv ([\mathsf{t}],\ \mathsf{id_c} \rightharpoonup [\bar{\mathsf{v}}]); \\ \bar{\mathsf{ms}} \equiv \mathsf{id_m} \rightharpoonup \bar{\mathsf{m}}; \\ \bar{\mathsf{s}} \equiv (\mathsf{cs},\ \mathsf{bs},\ \bar{\mathsf{ms}}); \end{array}$$

- In the $\mathbf{sell}^{\mathsf{ok}}$ rule of Section 3.2.2 the list of n tokens $\mathsf{vs}_\mathsf{c}^\mathsf{c}$ is compressed into a singleton list. This is achieved by replacing the function application $\mathsf{take}(\mathsf{n}, \mathsf{vs}_\mathsf{c}^\mathsf{c})$ by $[\mathsf{compress}(\mathsf{take}(\mathsf{n}, \mathsf{vs}_\mathsf{c}^\mathsf{c}))]$.

- The $\mathbf{clear}^{\mathsf{ok}}$ rule of Section 3.2.6 has to expand out the compressed tokens. Therefore, we replace the premise $\vdash \mathsf{vs}_\mathsf{c}^\mathsf{m} = \mathsf{vs}_\mathsf{c}^{\mathsf{m}'}$ by $\vdash \mathsf{expand}\ \mathsf{vs}_\mathsf{c}^\mathsf{m} = \mathsf{vs}_\mathsf{c}^{\mathsf{m}'}$.

Note that the definition of expand has been extended to lists of compressed tokens $[\bar{\mathsf{v}}]$; we will also extend the definition to $\bar{\mathsf{m}}$, $\bar{\mathsf{ms}}$ and $\bar{\mathsf{s}}$ for use in the proof later.

The changes described above give rise to a new relation $\overset{\bar{qp}}{\Rightarrow}$:

$$\overset{\bar{qp}}{\Rightarrow} :: \langle [\mathsf{tr}],\ \bar{\mathsf{s}} \rangle \leftrightarrow \langle [\mathsf{tr}],\ \bar{\mathsf{s}} \rangle;$$

In the appendix we prove that the optimised relation $\overset{\bar{qp}}{\Rightarrow}$ is a data refinement of the unoptimised relation $\overset{qp}{\Rightarrow}$.

5.4 An efficient solution

In practice it is difficult to compress random data effectively. To keep the cost of transaction down basically two options are available. Both compromise the security of the QuickPay scheme to an extent:

- Sacrifice some of the 'randomness' of the tokens. This would allow customers to 'guess' value tokens and thus to defraud the *broker*.

- Use lossy compression. This, as we will show below, allows the customer to defraud the *merchant*.

It is interesting to note how each of the alternatives creates problems for a different party.

We have experimented with lossy compression functions, whilst trying to avoid the extreme of the selecting lossy compression described earlier. Lossy compression functions do not have an inverse, so that the correctness proof no longer holds.

Our proposed lossy compression function adds the tokens vs. To expand the tokens, the broker uses the customers carnet to add the next n tokens and to check that the sum equals the compressed token.

$$\begin{array}{l} \bar{\mathsf{v}} \quad \equiv (\mathsf{n},\ \mathsf{v}); \\ \mathsf{compress\ vs} = (\#\mathsf{vs},\ \Sigma\ \mathsf{vs}); \end{array}$$

The following subsection shows some problems that might occur with the adding lossy compression. The first scenario is pessimistic and clears too few tokens, the second is optimistic and clears too many.

5.4.1 Scenario 6: pessimistic optimisation

The scenario below makes three **sell** transactions and then attempts to clear the tokens gathered all at once. The 'random' tokens produced by the broken are not random at all, but the carefully chosen sequence $[0, 1, 2, 3, 6, 4, 5, 7, 8, 9]$. The first three tokens are used for authentication purposes, and are not important for the scenario. However, the following pair of tokens $(3, 6)$ adds up to 9, and so does the next pair $(4, 5)$. During clearing, the sum token 9 is expanded into 3 and 6, although it was created from 4 and 5. Therefore, the two other tokens cannot be expanded and the whole transaction fails. It would have succeeded in the original, unoptimised protocol.

bank1 [0 1 2 3 6 4 5 7 8 9]

$\rightarrow \mathsf{stock}(\mathsf{shopx}, \mathsf{bank1}, 3) \rightarrow$

bank1 [3 6 4 5 7 8 9]
shopx ([0 1 2],0)
shopx [0 1 2]

$\rightarrow$ update(alice, bank1, 4) $\rightarrow$

alice [3 6 4 5]
bank1 [7 8 9]
alice ([3 6 4 5],96)
shopx ([0 1 2],0)

$\rightarrow$ sell(shopx, alice, 1) $\rightarrow$

alice [6 4 5]
shopx [0 1 2]
alice [1,3]

$\rightarrow$ sell(shopx, alice, 1) $\rightarrow$

alice [4 5]
shopx [0 1 2]
alice [1,6 1,3]

$\rightarrow$ sell(shopx, alice, 2) $\rightarrow$

shopx [0 1 2]
alice [2,9 1,6 1,3]

$\rightarrow$ auth_m(shopx, bank1) $\rightarrow$

bank1 [7 8 9]
alice ([3 6 4 5],96)
shopx ([1 2],0)
shopx [1 2]
alice [2,9 1,6 1,3]

$\rightarrow$ auth_b(bank1, shopx) $\rightarrow$

bank1 [7 8 9]
alice ([3 6 4 5],96)
shopx ([2],0)
shopx [2]
alice [2,9 1,6 1,3]

$\rightarrow$ clear(bank1, shopx, alice) $\rightarrow$
At this point the compressed token 9, which represents two uncompressed tokens, is expanded into $[3, 6]$.

5.4.2 Scenario 7: optimistic optimisation

The scenario below uses the same set of initial tokens as above. This time the customer spends tokens 3 and 6, and then refreshes her carnet (without increasing the value). She receives tokens 4 and 5. When the merchant then clears the sum token 9, everything appears to be in order, because the customers present carnet offers two tokens that also happen to add up to 9. In the unoptimised protocol the clearing would have failed.

bank1 [0 1 2 3 6 4 5 7 8 9]

$\rightarrow$ stock(shopx, bank1, 3) $\rightarrow$

bank1 [3 6 4 5 7 8 9]
shopx ([0 1 2],0)
shopx [0 1 2]

$\rightarrow$ update(alice, bank1, 2) $\rightarrow$

alice [3 6]
bank1 [4 5 7 8 9]
alice ([3 6],98)
shopx ([0 1 2],0)

$\rightarrow$ sell(shopx, alice, 2) $\rightarrow$

shopx [0 1 2]
alice [2,9]

$\rightarrow$ update(alice, bank1, 0) $\rightarrow$

alice [4 5]
bank1 [7 8 9]
alice ([4 5],98)
shopx ([0 1 2],0)

$\rightarrow$ auth_m(shopx, bank1) $\rightarrow$

bank1 [7 8 9]
alice ([4 5],98)
shopx ([1 2],0)
shopx [1 2]
alice [2,9]

$\rightarrow$ auth_b(bank1, shopx) $\rightarrow$

bank1 [7 8 9]
alice ([4 5],98)
shopx ([2],0)
shopx [2]
alice [2,9]

$\rightarrow$ clear(bank1, shopx, alice) $\rightarrow$

bank1 [7 8 9]
shopx ([2],2)
shopx [2]

Here the compressed token 9 is expanded into $[4, 5]$, causing the clearing to succeed.

6 Conclusions and future work

A formal model of the QuickPay micro payment system has been built that makes it possible:

- to clearly and concisely describe the transactions of the system. The core transactions of the protocol have been fully specified.

- to animate sample transaction sequences so as to illustrate the concepts and to explore scenarios of incorrect uses of the system. Seven such sequences have been presented.
- to identify potential problems and to study possible solutions to these problems. A new problem has been identified and an old problem (selecting lossy compression) has been re-discovered. Various solutions to the old problem are given, ranging from a provably correct solution (lossless compression) to an efficient solution (adding lossy compression).

The model has helped us to analyse the behaviour of the protocols, leading to the conclusion that QuickPay is:

- attractive for the customer. Even if she looses her tokens, they will be promptly replaced.
- attractive for the broker. Like all pre-paid schemes, the broker is able to dispose of the (real) money of the customer for a certain period of time.
- less attractive for the merchant. All problems that we have studied are to the disadvantage of the merchant. However, presently a merchant providing electronic services receives voluntary contributions at best. With QuickPay the merchant might expect an increase in revenue.

The model has been formulated rather abstractly, so that in the prototype implementation there may well be problems that are not captured by the model. The model could be extended to cover more detail.

The model could also be extended to study the costs of the transactions and to compare this costs to that of the tokens being processed.

The model could be extended to capture histories of customer and merchant behaviour. Such histories could be used to decide if and when clearing of tokens is needed, as well as other possible optimisations to the protocol. The histories could also be used to assess to what extent the merchant may take advantage of the knowledge of customer behaviour.

Finally, the model could be generalised to capture other micro payment systems, such as Mini-Pay, Millicent and Payword/Micromint. Such a general model would enable different micro payment systems to be compared on the same formal footing.

Acknowledgements

This work was supported by a short-term research fellowship from BT laboratories, Martlesham Heath, UK, contract number STRF97/33. The help and support of Michael Butler, John Regnault and Tim Hart is gratefully acknowledged.

References

[1] UK BT Research laboratories, Martlesham Heath. Transaction system. In *GB Patent application no. 9624127.8*, 1996.

[2] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. G. Sobalvarro. The millicent protocol for inexpensive electronic commerce. In *4th International World Wide Web Conf.*, pages 603–618, Boston, Massachusetts, Dec 1995. World Wide Web Journal. `www.research.digital.com/ SRC/ staff/ msm/ bio.html`.

[3] C. A. Gunter, I. Lee, and A. Scedrov. Two weak links in the formal methods chain. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*. Rutgers University, New Jersey, Sep 1997. `http:// dimacs.rutgers.edu/ Workshops/ Security/ program2/ program.html`.

[4] P. H. Hartel. LATOS – a lightweight animation tool for operational semantics. Technical report DSSE-TR-97-1, Dept. of Electr. and Comp. Sci, Univ. of Southampton, England, Oct 1997. `www.ecs.soton.ac.uk/ ~phh/ latos.html`.

[5] A. Herzberg and H. Yochai. Mini-Pay : Charging per click on the web. In *6th WWW conference*, Santa Clara, California, Apr 1997.

[6] J. Hill and M. Sims. QuickPay: Prepaid micropayments. Technical report in preparation, BT Research laboratories, Martlesham Heath, UK, 1998.

[7] D. O'Mahony, M. Peirce, and H. Tewari. *Electronic payment systems*. Artech House Inc, Boston, Massachusetts, 1997.

[8] R. L. Rivest and A. Shamir. *PayWord and MicroMint: Two simple micropayment schemes*. MIT, Cambridge, Massachusetts, May 1996. `http:// theory.lcs.mit.edu/ ~rivest/` .

[9] P. Ryan and I. Zakiuddin. Modelling and analysis of security protocols. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*. Rutgers University, New Jersey, Sep 1997. `http:// dimacs.rutgers.edu/ Workshops/ Security/ program2/ program.html`.

[10] B. Schneier. *Applied cryptography*. John Wiley & Sons, Chichester, England, second edition edition, 1996.

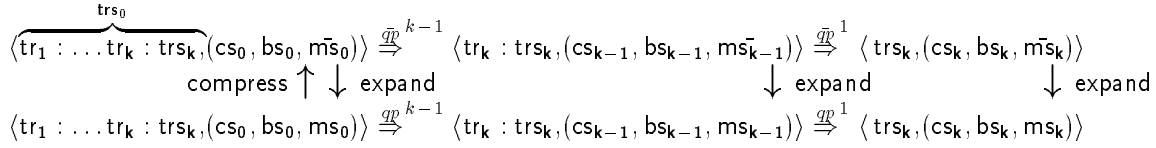

Figure 3: Inductive proof structure.

A Correctness of compression

To prove the correctness of lossless compression, we wish to assert that the initial and final configurations of the optimised and unoptimised protocols are the same, modulo compression/expansion. This assertion is formalised as follows:

Lemma

For all transaction sequences trs_0, states $(\mathsf{cs}_0, \mathsf{bs}_0, \mathsf{ms}_0)$, compression and decompression functions satisfying $\mathsf{expand}.\mathsf{compress} = \mathsf{id}$ and $0 \leq \mathsf{k} \leq \#\mathsf{trs}_0$ we have:

$$\begin{array}{l} \langle \mathsf{trs}_0, (\mathsf{cs}_0, \mathsf{bs}_0, \mathsf{compress}\ \mathsf{ms}_0) \rangle \overset{\bar{qp}}{\Rightarrow}^{k} \langle \mathsf{trs}_k, (\mathsf{cs}_k, \mathsf{bs}_k, \bar{\mathsf{ms}}_k) \rangle \\ \sqsubseteq \\ \langle \mathsf{trs}_0, (\mathsf{cs}_0, \mathsf{bs}_0, \mathsf{ms}_0) \rangle \overset{qp}{\Rightarrow}^{k} \langle \mathsf{trs}_k, (\mathsf{cs}_k, \mathsf{bs}_k, \mathsf{expand}\ \bar{\mathsf{ms}}_k) \rangle \end{array}$$

Proof

The proof proceeds by induction on the number of steps k in the transaction sequence. Figure 3 illustrates the proof strategy: Start with an initial configuration $\langle \mathsf{trs}_0, (\mathsf{cs}_0, \mathsf{bs}_0, \bar{\mathsf{ms}}_0) \rangle$ and proceed towards a new configuration $\langle \mathsf{trs}_k, (\mathsf{cs}_k, \mathsf{bs}_k, \bar{\mathsf{ms}}_k) \rangle$ in k steps. Given that the compressed and expanded initial configurations are related by $\bar{\mathsf{ms}}_0 = \mathsf{compress}\ \mathsf{ms}_0$ then we wish to show that the final configurations are related by $\mathsf{expand}\ \bar{\mathsf{ms}}_k = \mathsf{ms}_k$.

The proof of the base case uses $\mathsf{expand}.\mathsf{compress}\ \mathsf{ms}_0 = \mathsf{ms}_0$, which follows immediately from the requirement that expand is the inverse of compress.

To prove the general case, we perform case analysis on tr_k. This means that we should look at each of the rules defining the relations $\overset{qp}{\Rightarrow}$ and $\overset{\bar{qp}}{\Rightarrow}$, and prove that from the induction hypothesis $\mathsf{ms}_{k-1} = \mathsf{expand}\ \bar{\mathsf{ms}}_{k-1}$ and the properties of the appropriate rule we also have $\mathsf{ms}_k = \mathsf{expand}\ \bar{\mathsf{ms}}_k$.

Firstly, assume that tr_k is a failed transaction, that is the side condition of the appropriate rule is false; then

$$\begin{array}{ll} & \mathsf{expand}\ \bar{\mathsf{ms}}_k \\ \equiv & \{\ \ldots^{\mathsf{nok}}\ \text{transactions preserve}\ \bar{\mathsf{ms}}\ \} \\ & \mathsf{expand}\ \bar{\mathsf{ms}}_{k-1} \\ \sqsubseteq & \{\ \text{induction hypothesis}\ \} \\ & \mathsf{ms}_{k-1} \\ \equiv & \{\ \ldots^{\mathsf{nok}}\ \text{transactions preserve}\ \mathsf{ms}\ \} \\ & \mathsf{ms}_k \end{array}$$

Secondly, consider a successful update transaction. If $\mathsf{tr}_k = \mathsf{Update}(\mathsf{id}_c, \mathsf{id}_b, \mathsf{n})$ then $\mathsf{Update}^{\mathsf{ok}}$ does not affect ms or $\bar{\mathsf{ms}}$, so the reasoning above as for the failed transaction applies here also.

Thirdly, let $\mathsf{tr}_k = \mathsf{Sell}(\mathsf{id}_m, \mathsf{id}_c, \mathsf{n})$. The transaction $\mathsf{Sell}^{\mathsf{ok}}$ both uses and changes the merchant data, therefore we must consider the actions of the rule in detail. Consider how the premises of the rule $\mathsf{Sell}^{\mathsf{ok}}$ relate $\bar{\mathsf{ms}}_{k-1}$ and $\bar{\mathsf{ms}}_k$:

$$\begin{array}{ll} & \bar{\mathsf{ms}}_{k-1}(\mathsf{id}_m) = (\mathsf{ts}_m, \bar{\mathsf{cs}}_m) \\ & \bar{\mathsf{cs}}_m(\mathsf{id}_c) = \bar{\mathsf{vs}}_c^m \\ & \bar{\mathsf{cs}}_m \oplus \{\mathsf{id}_c \mapsto [\mathsf{compress}(\mathsf{take}(\mathsf{n}, \mathsf{vs}_c^c))] +\!\!+ \bar{\mathsf{vs}}_c^m\} = \bar{\mathsf{cs}}'_m \\ & (\mathsf{ts}_m, \bar{\mathsf{cs}}'_m) = \bar{\mathsf{m}}' \\ & \bar{\mathsf{ms}}_{k-1} \oplus \{\mathsf{id}_m \mapsto \bar{\mathsf{m}}'\} = \bar{\mathsf{ms}}_k \\ \equiv & \{\ \text{extension of expand c.f. Section 5.3}\ \} \\ & (\mathsf{expand}\ \bar{\mathsf{ms}}_{k-1})(\mathsf{id}_m) = (\mathsf{ts}_m, \mathsf{expand}\ \bar{\mathsf{cs}}_m) \\ & (\mathsf{expand}\ \bar{\mathsf{cs}}_m)(\mathsf{id}_c) = \mathsf{expand}\ \bar{\mathsf{vs}}_c^m \\ & (\mathsf{expand}\ \bar{\mathsf{cs}}_m) \oplus \{\mathsf{id}_c \mapsto [\mathsf{expand}(\mathsf{compress}(\mathsf{take}(\mathsf{n}, \mathsf{vs}_c^c)))] \\ & \quad +\!\!+ \mathsf{expand}\ \bar{\mathsf{vs}}_c^m\} = \mathsf{expand}\ \bar{\mathsf{cs}}'_m \\ & (\mathsf{ts}_m, \mathsf{expand}\ \bar{\mathsf{cs}}'_m) = \mathsf{expand}\ \bar{\mathsf{m}}' \\ & (\mathsf{expand}\ \bar{\mathsf{ms}}_{k-1}) \oplus \{\mathsf{id}_m \mapsto \mathsf{expand}\ \bar{\mathsf{m}}'\} = \mathsf{expand}\ \bar{\mathsf{ms}}_k \\ \sqsubseteq & \{\ \text{hypothesis, expand.compress} = \mathsf{id}\ \} \\ & \mathsf{ms}_{k-1}(\mathsf{id}_m) = (\mathsf{ts}_m, \mathsf{cs}_m) \\ & \mathsf{cs}_m(\mathsf{id}_c) = \mathsf{vs}_c^m \\ & \mathsf{cs}_m \oplus \{\mathsf{id}_c \mapsto (\mathsf{take}(\mathsf{n}, \mathsf{vs}_c^c) +\!\!+ \mathsf{vs}_c^m)\} = \mathsf{cs}'_m \\ & (\mathsf{ts}_m, \mathsf{cs}'_m) = \mathsf{m}' \\ & \mathsf{ms}_{k-1} \oplus \{\mathsf{id}_m \mapsto \mathsf{m}'\} = \mathsf{ms}_k \end{array}$$

Fourthly, let $\mathsf{tr}_k = \mathsf{Clear}(\mathsf{id}_b, \mathsf{id}_m, \mathsf{id}_c)$. The transaction $\mathsf{Clear}^{\mathsf{ok}}$ also uses and changes the merchant record. Studying how the premises relate $\bar{\mathsf{ms}}_{k-1}$ and $\bar{\mathsf{ms}}_k$ yields:

$$
\begin{array}{ll}
 & \bar{ms}_{k-1}\{id_m\} = (ts^m_m, \bar{cs}_m) \\
 & \bar{cs}_m\{id_c\} = \bar{vs}^m_c \\
 & \text{expand } \bar{vs}^m_c = vs^{m'}_c \\
 & (ts^m_m, \bar{cs}_m \oplus \{id_c \mapsto []\}) = \bar{m}' \\
 & \bar{ms}_{k-1} \oplus \{id_m \mapsto \bar{m}'\} = \bar{ms}_k \\
\equiv & \quad \{ \text{ extension of expand } \} \\
 & (\text{expand } \bar{ms}_{k-1})\{id_m\} = (ts^m_m, \text{expand } \bar{cs}_m) \\
 & (\text{expand } \bar{cs}_m)\{id_c\} = \text{expand } \bar{vs}^m_c \\
 & \text{expand } \bar{vs}^m_c = vs^{m'}_c \\
 & (ts^m_m, \text{expand } \bar{cs}_m \oplus \{id_c \mapsto []\}) = \text{expand } \bar{m}' \\
 & (\text{expand } \bar{ms}_{k-1}) \oplus \{id_m \mapsto \bar{m}'\} = \text{expand } \bar{ms}_k \\
\sqsubseteq & \quad \{ \text{ hypothesis, extension of expand } \} \\
 & ms_{k-1}\{id_m\} = (ts^m_m, cs_m) \\
 & cs_m\{id_c\} = vs^m_c \\
 & vs^m_c = vs^{m'}_c \\
 & (ts^m_m, cs_m \oplus \{id_c \mapsto []\}) = m' \\
 & ms_{k-1} \oplus \{id_m \mapsto m'\} = ms_k
\end{array}
$$

Finally, the remaining rules $\mathbf{stock}^{ok}$, $\mathbf{auth}^{ok}_m$ and $\mathbf{auth}^{ok}_b$ do not actually use or change the customer tokens in the merchant records, which completes the proof.