# Formal Analysis of the *i*KP Electronic Payment Protocols

Kazuhiro Ogata and Kokichi Futatsugi

August 20, 2002

IS-RR-2002-020

# Formal Analysis of the $i$KP Electronic Payment Protocols

Kazuhiro Ogata
NEC Software Hokuriku, Ltd.
and
Japan Advanced Institute of Science and Technology (JAIST)
ogatak@acm.org

Kokichi Futatsugi
Graduate School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
kokichi@jaist.ac.jp

## Abstract

*$i$KP ($i$-Key-Protocol, $i = 1, 2, 3$) is a family of electronic payment protocols, developed in early 1995 by a group of researchers at the IBM Research labs in Yorktown Heights and Zürich, and one of the ancestors of well-known SET standard. In this paper, we analyze $i$KP with respect that they have a property that buyer and seller always agreed on the payment whenever acquirer authorizes it. As the designers of the $i$KP protocols point out, the 1KP protocol does not possess the property. We found out, however, that there exists a counter example to the 2KP and 3KP protocols. Threfore we propose modification of the 2KP and 3KP protocols so that they can possess the property. We have formally verified that the modified 2KP and 3KP protocols possess the property. In this paper, we describe the verification that the modified 3KP protocol possesses the property.*

**Keywords:** CafeOBJ, electronic commerce, the $i$KP electronic payment protocols, observational transition systems, proof scores, rewriting, verification.

## 1. Introduction

Nobody doubts that security protocols are a key to success of sound development of the Internet, especially success of electronic commerce. But, they are subject to subtle errors that are especially difficult to reveal by traditional testing methods and usual operations. Actually Lowe[15] found out a serious security flaw of the Needham-Schroeder Public-Key authentication protocol[21], or the NSPK protocol 17 years later since the protocol had been proposed. This demonstrates that errors lurked in security protocols are very subtle, and has motivated many researchers to apply formal methods to security protocols so as to analyze them.

$i$KP ($i$-Key-Protocol, $i = 1, 2, 3$)[4, 3] is a family of electronic payment protocols, developed in early 1995 by a group of researchers at the IBM Research labs in Yorktown Heights and Zürich. They have affected the design of well-known SET standard[20]. In this paper, we analyze $i$KP with respect that they have a property that buyer and seller always agree on the payment whenever acquirer authorizes it, which is called agreement property in this paper. As the designers of the $i$KP protocols point out, the 1KP protocol does not have the property. We found out, however, that there exists a counter example to the 2KP and 3KP protocols. Threfore we propose modification of the 2KP and 3KP protocols so that they can possess the property. We have formally verified that the modified 2KP and 3KP protocols possess the property. In this paper, we describe the verification that the modified 3KP protocol possesses the property.

The verification has been done with CafeOBJ[5, 9]. CafeOBJ is an algebraic specification language in which abstract machines or objects in object-orientation as well as abstract data types can be described. The verification process is roughly as follows. First the modified 2KP and 3KP protocol have been abstracted to ease the verification, which are called the AM2KP and AM3KP protocols. Next each of the AM2KP and AM3KP protocols has been modeled as an observational transition system (an OTS)[23, 22, 24] and the OTS has been described in CafeOBJ. Then proof scores to show that the AM2KP and AM3KP protocols possess agreement property have been written in CafeOBJ and have got executed by the CafeOBJ system. Writing proof scores in algebraic specification languages was first advocated by Goguen's group and developed for more than 15 years in

OBJ community[11, 13]. This paper also shows that the approach can be applied to analyzing security protocols.

The rest of the paper is organized as follows. Section 2 provides a summary of the $i$KP electronic payment protocols. Section 3 defines agreement property and shows some counter examples with respect to the property. We propose modificatoin of the 2KP and 3KP protocols so that they can possess the property in Sect. 4. In Sect. 5, we describe how to model the modified 3KP protocol. In the section, we first abstract the modified $i$KP protocols to ease the verification, which are called the AM$i$KP protocols. We next write observational transition systems (OTS's) and how to describe OTS's in CafeOBJ. We finally describe the OTS modeling the AM3KP protocol and its CafeOBJ document in the section. Section 6 describes the verification that the AM3KP protocol possesses agreement property. Section 7 mentions the related work, and finally we conclude with Sect. 8.

## 2. The $i$KP Electronic Payment Protocols

$i$KP ($i$-Key-Protocol, $i = 1, 2, 3$)[4, 3] is a family of electronic payment protocols, developed in early 1995 by a group of researchers at the IBM Research labs in Yorktown Heights and Zürich. Afterward it was incorporated into the "Secure Electronic Payment Protocols (SEPP)," a short-lived standardization effort by IBM, MasterCard, Europay and Netscape. SEPP, in turn, was a key starting point for "Secure Electronic Payments (SET)," the joint VISA/MasterCard standard for credit card payments[20]. In fact, SET will retains many of the $i$KP-esque features.

All $i$KP protocols are based on the existing credit-card payment system. The parties in the payment system are show in Fig. 1. The $i$KP protocols deal with the payment transaction only (namely the solid lines in Fig. 1) and therefore involve only three parties called $B$ (Buyer), $S$ (Seller) and $A$ (Acquirer). Note that $A$ is not the acquirer in the financial sense, but a gateway to the existing credit card clearing/authorization network.

The payment system is operated by a payment system provider who maintains a fixed business relationship with a number of banks. Banks act as credit card (account) issuer to buyers, and/or as acquirers of payment records from merchants (sellers). It is assumed that each buyer receives its credit card from an issuer, and is somehow assigned (or selects) an optional PIN as its common in current credit card systems. In 1KP and 2KP, payments are authorized only by means of the credit card number and the optional PIN (both suitably encrypted), while, in 3KP, a digital signature is used, in addition to the above. A seller signs up with the payment system provider and with a specific bank, called an acquirer, to accept deposits. Clearing between acquirers and issuers is done using the existing financial networks.

All $i$KP protocols are based on public key cryptography, and each acquirer $A$ has a secret key $\mathrm{SK}_A$ that enables signing and decryption. In this paper, for brevity, we assume that its public counterpart $\mathrm{PK}_A$ that enables signature verification and encryption is securely conveyed to every buyer and seller participating the protocols via any of a number of key distribution mechanisms. Each seller $S$ in 2KP/3KP and each buyer $B$ in 3KP has a secret/public key-pair $(\mathrm{SK}_S, \mathrm{PK}_S)$ and $(\mathrm{SK}_S, \mathrm{PK}_S)$, respectively. We also assume that each seller's public key is securely conveyed to every acquirer and buyer in 2KP/3KP, and that each buyer's public key is securely conveyed to every acquirer and seller in 3KP.

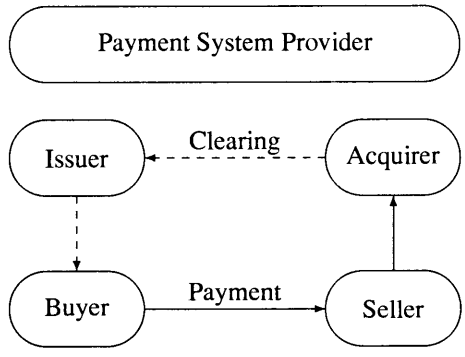


**Figure 1. Generic model of a payment system**

Cryptographic primitives used in the protocols are as follows:

- $\mathcal{H}(\cdot)$ : A strong collision-resistant one-way hash function that returns strong pseudo-random values.
- $\mathcal{H}_k(K, \cdot)$ : A one-way hash function requiring in addition to collision-resistance, no information leakage with respect to its other arguments, if the first argument $K$ is chosen at random.
- $\mathcal{E}_X(\cdot)$ : Public-key encryption with $\mathrm{PK}_X$, performed in a way to provide both confidentiality and some kind of computational message integrity.
- $\mathcal{S}_X(\cdot)$ : Signature computed with $\mathrm{SK}_X$.

Figure 2 shows the three $i$KP protocols. Parts enclosed by $[_{2,3}\ldots]$ and $[_3\ldots]$ are ignored for 1KP and 2KP respectively. The main difference between 1, 2 and 3KP is the increasing use of digital signatures as more of the parties involved possess a public/secret key pair.

Quantities occurring in the protocols are as follows:

- $\mathrm{SALT}_B$ : Random number generated by $B$. Used to salt DESC and thus ensure privacy of order information (DESC) on the $S$ to $A$ link; also used to provide freshness of signatures.

**Composite fields:**

| | |
|---|---|
| Common | $\text{AUTHPRICE}, \text{ID}_S, \text{TID}_S, \text{DATE}, \text{NONCE}_S, \text{ID}_B, \mathcal{H}_k(\text{SALT}_B, \text{DESC}), [_{2,3}\mathcal{H}(\text{V}), \mathcal{H}(\text{VC})]$ |
| Clear | $\text{ID}_S, \text{TID}_S, \text{DATE}, \text{NONCE}_S, \mathcal{H}(\text{Common}), [_{2,3}\mathcal{H}(\text{V}), \mathcal{H}(\text{VC})]$ |
| SLIP | $\text{AUTHPRICE}, \mathcal{H}(\text{Common}), \text{BAN}, \text{R}_B, [\text{PIN}], \text{EXPIRATION}$ |
| EncSlip | $\mathcal{E}_A(\text{SLIP})$ |
| $\text{Sig}_A$ | $\mathcal{S}_A(\text{RESPCODE}, \mathcal{H}(\text{Common}))$ |
| $\text{Sig}_S$ | $\mathcal{S}_S(\mathcal{H}(\text{Common}))$ |
| $\text{Sig}_B$ | $\mathcal{S}_B(\text{EncSlip}, \mathcal{H}(\text{Common})))$ |

**Starting information of parties:**

| | |
|---|---|
| $B$ | $\text{DESC}, \text{AUTHPRICE}, \text{BAN}, \text{EXPIRATION}, \text{PK}_A, [\text{PIN}], [_{2,3}\text{PK}_S, [_3\text{SK}_B]]$ |
| $S$ | $\text{DESC}, \text{AUTHPRICE}, \text{PK}_A, [_{2,3}\text{SK}_S, [_3\text{PK}_B]]$ |
| $A$ | $\text{SK}_A, \text{PK}_A, [_{2,3}\text{PK}_S, [_3\text{PK}_B]]$ |

**Protocol flows:**

| | | | | |
|---|---|---|---|---|
| Initiate: | $B$ | $\longrightarrow$ | $S$ | $: \text{SALT}_B, \text{ID}_B$ |
| Invoice: | $S$ | $\longrightarrow$ | $B$ | $: \text{Clear}, [_{2,3}\text{Sig}_S]$ |
| Payment: | $B$ | $\longrightarrow$ | $S$ | $: \text{EncSlip}, [_3\text{Sig}_B]$ |
| Auth-Request: | $S$ | $\longrightarrow$ | $A$ | $: \text{Clear}, \mathcal{H}_k(\text{SALT}_B, \text{DESC}), \text{EncSlip}, [_{2,3}\text{Sig}_S, [_3\text{Sig}_B]]$ |
| Auth-Response: | $A$ | $\longrightarrow$ | $S$ | $: \text{RESPCODE}, \text{Sig}_A$ |
| Confirm: | $S$ | $\longrightarrow$ | $B$ | $: \text{RESPCODE}, \text{Sig}_A, [_{2,3}\text{V}\|\text{VC}]$ |

**Figure 2. *i*KP protocols**

- AUTHPRICE : Amount and currency.
- DATE : Seller's date/time stamp, used for coarse-grained payment replay protection.
- $\text{NONCE}_S$ : Seller's nonce (random number) used for more fine-grained payment replay protection.
- $\text{ID}_S$ : Seller ID.
- $\text{TID}_S$ : Transaction ID.
- DESC : Description of purchase/goods, and delivery address. Includes payment information such as credit card name, bank identification number, and currency. Defines agreement between buyer and seller as to what is being paid for in this payment transaction.
- BAN : Buyer's Account Number such as credit card number.
- EXPIRATION : Expiration date associated with Buyer's Account Number.
- $\text{R}_B$ : Random number chosen by buyer to form $\text{ID}_B$. It must be random in order to serve as proof to the buyer that the seller agreed to the payment.
- $\text{ID}_B$ : A buyer pseudo-ID computed as $\text{ID}_B = \mathcal{H}(\text{R}_B, \text{BAN})$.
- RESPCODE : Response from the clearing network: YES/NO or authorization code.
- PIN : Buyer PIN.
- V : Random number generated by seller in 2KP/3KP for use as a proof that seller has accepted payment.
- VC : Random number generated by seller in 2KP/3KP for use as a proof that seller has not accepted payment.

We are about to describe how the *i*KP protocols work. Before each protocol starts, each party has the information as shown in Fig. 2. Each buyer $B$ has an account number BAN and associated EXPIRATION, both known to the payment system. $B$ may also have a secret PIN that is also known (possibly under a one-way function image) to the payment system.

Initiate: Buyer forms $\text{ID}_B$ by generating a random number $\text{R}_B$ and computing $\text{ID}_B = \mathcal{H}_k(\text{R}_B, \text{BAN})$. Buyer generates another random number $\text{SALT}_B$ to be used for salt-

ing the hash of merchandise description (DESC) in subsequent flows. Buyer sends Initiate flow.

Invoice: Seller retrieves $\mathrm{SALT}_B$ and $\mathrm{ID}_B$ from Initiate, and obtains DATE and generates a random quantity $\mathrm{NONCE}_S$. The combination of DATE and $\mathrm{NONCE}_S$ is used later by $A$ to uniquely identify this payment. Seller then chooses a transaction ID $\mathrm{TID}_S$ that identifies the context and computes $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$. In 2KP/3KP, seller also generates two random values V and VC, and then computes $\mathcal{H}(\mathrm{V})$ and $\mathcal{H}(\mathrm{VC})$. Seller forms Common as defined above and computes $\mathcal{H}(\mathrm{Common})$. In 2KP/3KP, seller also computes $\mathrm{Sig}_S$ $(= \mathcal{S}_S(\mathcal{H}(\mathrm{Common})))$. Finally seller sends Invoice.

Payment: Buyer retrieves Clear from Invoice and validates DATE within a pre-defined time skew. Buyer computes $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$ and $\mathcal{H}(\mathrm{Common})$, and checks it matches the corresponding value in Clear. In 2KP/3KP, buyer also validates the signature retrieved from Invoice using $\mathrm{PK}_S$. Next buyer forms SLIP as defined in Fig. 2 and encrypts it using $\mathrm{PK}_A$ (EncSlip $= \mathcal{E}_A(\mathrm{SLIP})$). In 3KP, buyer also computes $\mathrm{Sig}_B$ $(= \mathcal{S}_B(\mathrm{EncSlip}, \mathcal{H}(\mathrm{Common})))$. Finally buyer sends Payment.

Auth-Request: In 3KP, seller validates the signature retrieved from Payment using $\mathrm{PK}_B$. Seller forwards EncSlip (and also $\mathrm{Sig}_B$ in 3KP) along with Clear and $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$ (and also $\mathrm{Sig}_S$ in 2KP/3KP) as Auth-Request.

Auth-Response: Acquirer extracts Clear, EncSlip and $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$ (and also $\mathrm{Sig}_S$ in 2KP/3KP and furthermore $\mathrm{Sig}_B$ in 3KP) from Auth-Request. Acquirer then does the following:

1. Extracts $\mathrm{ID}_S$, $\mathrm{TID}_S$, DATE, $\mathrm{NONCE}_S$ and the value $h_1$ presumably corresponding to $\mathcal{H}(\mathrm{Common})$ from Clear. In 2KP/3KP, also extracts $\mathcal{H}(\mathrm{V})$ and $\mathcal{H}(\mathrm{VC})$. Acquirer checks for replays, namely makes sure that there is no previously processed request with the same quadruple $(\mathrm{ID}_S, \mathrm{TID}_S, \mathrm{DATE}, \mathrm{NONCE}_S)$.

2. Decrypts EncSlip. If decryption fails, acquirer assumes that EncSlip has been altered and the transaction is therefore invalid. Otherwise, acquirer obtains SLIP and, from it, extracts AUTHPRICE, $h_2$ (corresponding to $\mathcal{H}(\mathrm{Common})$), BAN, EXPIRATION, $\mathrm{R}_B$ and optionally PIN.

3. Checks that $h_1$ and $h_2$ match.

4. Reconstructs Common, computes $\mathcal{H}(\mathrm{Common})$ and checks that it matches $h_1$.

5. In 2KP/3KP, validates $\mathrm{Sig}_S$ using $\mathrm{PK}_S$.

6. In 3KP, validates $\mathrm{Sig}_B$ using $\mathrm{PK}_B$.

7. Uses the credit card organization's existing clearing and authorization system to obtain on-line authorization of this payment. This entails forwarding BAN, EXPIRATION, PIN (if present), price, etc. as dictated by the authorization system. Upon receipt of a response RESPCODE from the authorization system, acquirer computes a signature on RESPCODE and $\mathcal{H}(\mathrm{Common})$.

Finally acquirer sends Auth-Response to seller.

Confirm: Seller extracts RESPCODE and the acquirer's signature from Auth-Response. Seller then validates the signature using $\mathrm{PK}_A$ and forwards both RESPCODE and the signature as Confirm. In 2KP/3KP, either V or VC is also included in Confirm depending on RESPCODE.

## 3. Agreement Property

There are several properties that electronic payment protocols such as the $i$KP protocols must have. For example they must make it impossible for intruders or malicious sellers to launch replay attack. The property that we deal with in this paper is as follows:

> *Buyer and seller always agreed on the payment whenever acquirer authorizes it.*

The property is called *agreement property* in this paper.

In the $i$KP protocols, acquirer must receive valid Auth-Request in the sense described in the previous section, no matter who has generated, so that she/he can have the existing authorization system check the payment. Moreover that buyer and seller agreed on the payment (namely the valid Auth-Request) can be stated as they have generated Initiate and Payment, and Invoice and Auth-Request corresponding to the valid Auth-Request respectively. Therefore agreement property can be restated as follows:

> *Involved buyer and seller have always generated* Initiate *and* Payment, *and* Invoice *and* Auth-Request *corresponding to the valid* Auth-Request *respectively whenever acquirer receives valid* Auth-Request, *no matter who has generated.*

Do all the $i$KP protocols have this property? The answer is NO!

**(1) In 1KP** Clear′ and EncSlip′ are Clear and EncSlip replaced AUTHPRICE with AUTHPRICE′ respectively.

| | | | | | |
|---|---|---|---|---|---|
| Initiate: | $IB$ | $\longrightarrow$ | $S$ | : | $\mathrm{SALT}_{IB}, \mathrm{ID}_{IB}$ |
| Invoice: | $S$ | $\longrightarrow$ | $IB$ | : | Clear |
| Payment: | $IB$ | $\longrightarrow$ | $S$ | : | EncSlip |
| Auth-Request: | $S$ | $\longrightarrow$ | $A$ | : | $\mathrm{Clear}, \mathcal{H}_k(\mathrm{SALT}_{IB}, \mathrm{DESC}), \mathrm{EncSlip}$ |
| Auth-Request′: | $IS(S)$ | $\longrightarrow$ | $A$ | : | $\mathrm{Clear}', \mathcal{H}_k(\mathrm{SALT}_{IB}, \mathrm{DESC}), \mathrm{EncSlip}'$ |
| Auth-Response: | $A$ | $\longrightarrow$ | $S$ | : | $\mathrm{RESPCODE}, \mathrm{Sig}_A$ |

**(2) In 3KP**

| | | | | | |
|---|---|---|---|---|---|
| Initiate: | $IB$ | $\longrightarrow$ | $S$ | : | $\mathrm{SALT}_{IB}, \mathrm{ID}_B$ |
| Invoice: | $S$ | $\longrightarrow$ | $IB$ | : | $\mathrm{Clear}, \mathrm{Sig}_S$ |
| Auth-Request: | $IS(S)$ | $\longrightarrow$ | $A$ | : | $\mathrm{Clear}, \mathcal{H}_k(\mathrm{SALT}_{IB}, \mathrm{DESC}), \mathrm{EncSlip}, \mathrm{Sig}_S, \mathrm{Sig}_{IB}$ |
| Auth-Response: | $A$ | $\longrightarrow$ | $S$ | : | $\mathrm{RESPCODE}, \mathrm{Sig}_A$ |

Suppose that there exists an intruder that is also a legitimate principal. $IB$ and $IS$ stand for the intruder acting as a buyer and a seller respectively. $IS(S)$ and $IB(B)$ mean that $IS$ and $IB$ impersonate $S$ and $B$ respectively.

**Figure 3. Counter examples**

As the designers of the $i$KP protocols point out, 1KP does no have the property. Although you can easily generate counter examples for 1KP, one of the interesting counter examples is shown in Fig. 3 (1). We assume that there exits an intruder that can also act as a legitimate principal in the protocols. The intruder can eavesdrop any message flowing in the network and, from it, glean any quantity except those cryptographically processed (namely it is assumed that the cryptsystem used cannot be broken). In the example shown in Fig. 3 (1), the intruder impersonates $S$ and sends Auth-Request′ to $A$ before $A$ receives Auth-Request from $S$. Since the intruder knows all the quantities to compute Auth-Request′, she/he can generate and send it to $A$, and then $A$ receives it as valid. If AUTHPRICE′ is smaller than AUTHPRICE, the payment would be disadvantageous to $S$. Although $S$ will notice that this payment transaction is not valid by checking $\mathrm{Sig}_A$ against RESPCODE and $\mathcal{H}(\mathrm{Common})$ computed by himself/herself, he/she cannot prove it invalid to others.

How about 2KP/3KP? They seemingly possess the property, but there exists an counter example shown in Fig. 3 (2). What advantage can the intruder get from the counter example? We can imagine several. $S$ might want to cancel $IB$'s payment request due to some reason if $S$ received Payment from $IB$, although the cancelation is outside the scope of the $i$KP protocols. In the counter example, $A$ accepts Auth-Request regardless of $S$'s intention.

The intruder just wants to confuse the payment system. $S$ receives Confirm from $A$ even if $S$ has never sent the corresponding Auth-Request to $A$, and gets aware that something, no matter what it is, that does not follow the protocol has been occurred. $S$ might decide not to use the payment system because $S$ cannot believe the payment system any more. Getting worse, the media cover this unexpected behavior of the payment system, and more people stop making use of the payment system. This is clearly disadvantageous to the payment system.

If possible, don't you think that electronic payment protocols must have agreement property? In the following sections, we propose a modification of the $i$KP protocols and formally verify that the modified 3KP protocol possesses the property.

## 4. Modification of the $i$KP Protocols

We propose two modifications to the $i$KP protocols.

The reason why the counter example shown in Fig. 3 (2) can be occurred is that $IB$ receives Invoice and gains all the quantities to generate valid Auth-Request. If $S$ newly makes another signature when it sends Auth-Request, not reusing $\mathrm{Sig}_S$ used for sending Invoice, then the counter example cannot be occurred. Threfore the first modification is making a different signature for sending Auth-Request than that used for sending Invoice.

The second modification is adding the involved buyer's ID into Common. This modification is not essential, but makes it possible to ease the verification described later.

Figure 4 shows the modified $i$KP protocols. In the modified $i$KP protocols, $S$ is used as seller's ID instead of $\mathrm{ID}_S$. The modification is as follows. $B$ as buyer's ID is added into Common. $\mathrm{Sig2}_S$ is newly introduced. It is used for sending Auth-Request instead of $\mathrm{Sig}_S$.

**Composite fields:**

| | |
|---|---|
| Common | AUTHPRICE, $S, B, \mathrm{TID}_S, \mathrm{DATE}, \mathrm{NONCE}_S, \mathrm{ID}_B, \mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC}), [_{2,3}\mathcal{H}(\mathrm{V}), \mathcal{H}(\mathrm{VC})]$ |
| Clear | $\mathrm{ID}_S, \mathrm{TID}_S, \mathrm{DATE}, \mathrm{NONCE}_S, \mathcal{H}(\mathrm{Common}), [_{2,3}\mathcal{H}(\mathrm{V}), \mathcal{H}(\mathrm{VC})]$ |
| SLIP | AUTHPRICE, $\mathcal{H}(\mathrm{Common}), \mathrm{BAN}, \mathrm{R}_B, [\mathrm{PIN}], \mathrm{EXPIRATION}$ |
| EncSlip | $\mathcal{E}_A(\mathrm{SLIP})$ |
| $\mathrm{Sig}_A$ | $\mathcal{S}_A(\mathrm{RESPCODE}, \mathcal{H}(\mathrm{Common}))$ |
| $\mathrm{Sig}_S$ | $\mathcal{S}_S(\mathcal{H}(\mathrm{Common}))$ |
| $\mathrm{Sig2}_S$ | $\mathcal{S}_S(\mathcal{H}(\mathrm{Common}),\mathrm{EncSlip})$ |
| $\mathrm{Sig}_B$ | $\mathcal{S}_B(\mathrm{EncSlip}, \mathcal{H}(\mathrm{Common}))$ |

**Protocol flows:**

| | | | | | |
|---|---|---|---|---|---|
| Initiate: | $B$ | $\longrightarrow$ | $S$ | : | $\mathrm{SALT}_B, \mathrm{ID}_B$ |
| Invoice: | $S$ | $\longrightarrow$ | $B$ | : | $\mathrm{Clear}, [_{2,3}\mathrm{Sig}_S]$ |
| Payment: | $B$ | $\longrightarrow$ | $S$ | : | $\mathrm{EncSlip}, [_3\mathrm{Sig}_B]$ |
| Auth-Request: | $S$ | $\longrightarrow$ | $A$ | : | $\mathrm{Clear}, \mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC}), \mathrm{EncSlip}, [_{2,3}\mathrm{Sig2}_S, [_3\mathrm{Sig}_B]]$ |
| Auth-Response: | $A$ | $\longrightarrow$ | $S$ | : | $\mathrm{RESPCODE}, \mathrm{Sig}_A$ |
| Confirm: | $S$ | $\longrightarrow$ | $B$ | : | $\mathrm{RESPCODE}, \mathrm{Sig}_A, [_{2,3}\mathrm{V}\|\mathrm{VC}]$ |

**Figure 4. Modified $i$KP protocols**

## 5. Modeling the Modified 3KP Protocol

First we abstract the modified $i$KP protocols so that we can reasonably verify that the modified 3KP protocol has agreement protocol. Next observational transition systems used for modeling the protocol and how to describe such models in CafeOBJ are written. Finally how to model the abstraction of the modified 3KP protocol as an observational transition system is written and its description in CafeOBJ is shown.

### 5.1. Abstraction of the Modified $i$KP Protocols

Since the (modified) $i$KP protocols should have other properties such as replay attack protection than agreement property, messages exchanged between principals includes information that is not directly needed to possess the property. Therefore we first abstract the modified $i$KP protocols so that we can reasonably verify that the modified 3KP protocol has the property.The abstraction is called the AM$i$KP protocols.

AUTHPRICE, $\mathcal{H}(\mathrm{V})$ and $\mathcal{H}(\mathrm{VC})$ are removed because they are most likely irrelevant to agreement property. Although $\mathrm{TID}_S$, DATE and $\mathrm{NONCE}_S$ are also used so that $S$ can remember which buyer $S$ is communicating with using Common including these quantities, they are removed because buyer's ID is included in Common. BAN, PIN and EXPIRATION are used so that acquirer can check that the credit card is valid. In the AM$i$KP protocols, only BAN is used and the remaining two are deleted. DESC is deleted because it seems irrelevant to agreement property, and $\mathrm{SALT}_B$ is removed because it is primarily used for salting DESC. Moreover Confirm is irrelevant to agreement property as well, and Confirm message flow is deleted.

Figure 5 shows the AM$i$KP protocols.

Since the abstraction done is simply deleting some quantities, if we verify that the AM3KP protocol has agreement property, then we are assured that the modified 3KP protocol really does.

### 5.2. Observational Transition Systems

**Models** We assume that there exists a universal state space called $\Upsilon$. You can imagine that an element, called a state, of $\Upsilon$ corresponds to a snapshot of the universe. When we describe a system, the system is basically modeled by observing only quantities that are relevant to the system and that interest us from the outside of each state of $\Upsilon$. An observational transition system[23, 24, 25], or an OTS can be used to model a system in this way.

An OTS $\mathcal{S} = \langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ consists of:

- $\mathcal{O}$: A set of observations. Each observation $o \in \mathcal{O}$ is a function $o : \Upsilon \rightarrow D$ mapping each $v \in \Upsilon$ into some typed value in $D$ ($D$ may be different for each observation). The value returned by an observation (in a state) is called the value of the observation (in the state).

**Composite fields:**

| | |
|---|---|
| Common | $S, B, \mathrm{ID}_B$ |
| Clear | $\mathcal{H}(\mathrm{Common})$ |
| SLIP | $\mathcal{H}(\mathrm{Common}), \mathrm{BAN}, \mathrm{R}_B$ |
| EncSlip | $\mathcal{E}_A(\mathrm{SLIP})$ |
| $\mathrm{Sig}_A$ | $\mathcal{S}_A(\mathrm{RESPCODE}, \mathcal{H}(\mathrm{Common}))$ |
| $\mathrm{Sig}_S$ | $\mathcal{S}_S(\mathcal{H}(\mathrm{Common}))$ |
| $\mathrm{Sig2}_S$ | $\mathcal{S}_S(\mathcal{H}(\mathrm{Common}), \mathrm{EncSlip})$ |
| $\mathrm{Sig}_B$ | $\mathcal{S}_B(\mathrm{EncSlip}, \mathcal{H}(\mathrm{Common}))$ |

**Protocol flows:**

| | | | | | |
|---|---|---|---|---|---|
| Initiate: | $B$ | $\longrightarrow$ | $S$ | : | $\mathrm{ID}_B$ |
| Invoice: | $S$ | $\longrightarrow$ | $B$ | : | $\mathrm{Clear}, [_{2,3}\mathrm{Sig}_S]$ |
| Payment: | $B$ | $\longrightarrow$ | $S$ | : | $\mathrm{EncSlip}, [_3\mathrm{Sig}_B]$ |
| Auth-Request: | $S$ | $\longrightarrow$ | $A$ | : | $\mathrm{Clear}, \mathrm{EncSlip}, [_{2,3}\mathrm{Sig2}_S, [_3\mathrm{Sig}_B]]$ |
| Auth-Response: | $A$ | $\longrightarrow$ | $S$ | : | $\mathrm{RESPCODE}, \mathrm{Sig}_A$ |

**Figure 5. The AM*i*KP protocols**

Given an OTS $\mathcal{S}$ and two states $v_1, v_2 \in \Upsilon$, the equality between two states, denoted by $v_1 =_{\mathcal{S}} v_2$, with respect to $\mathcal{S}$ is defined as follows:

$$v_1 =_{\mathcal{S}} v_2 \text{ iff } \forall o \in \mathcal{O}.o(v_1) = o(v_2),$$

where '=' in $o(v_1) = o(v_2)$ is supposed to be well defined for the range of each $o \in \mathcal{O}$. $\mathcal{S}$ may be removed from $=_{\mathcal{S}}$ if it is clear from the context.

- $\mathcal{I}$: The initial condition. This condition specifies the initial value of each observation that defines initial states of the OTS.
- $\mathcal{T}$: A set of conditional transition rules. Each transition rule $\tau \in \mathcal{T}$ is a relation between states provided that, for each state $v \in \Upsilon$, there exists a state $v' \in \Upsilon$, called a successor state, such that $\tau(v, v')$ and moreover, for each state $v_1, v_2, v_1', v_2' \in \Upsilon$ such that $v_1 =_{\mathcal{S}} v_2$, $\tau(v_1, v_1')$ and $\tau(v_2, v_2')$, $v_1' =_{\mathcal{S}} v_2'$. $\tau$ can be regarded as a function on equivalent classes of $\Upsilon$ with respect to $=_{\mathcal{S}}$. Therefore, we assume that $\tau(v)$ denotes the representative element of the equivalent class the successor states of $v$ with respect $\tau$ belong to.

  The condition $c_\tau$ for a transition rule $\tau \in \mathcal{T}$ is called the *effective condition*. Given a state, its truth value can be determined by only the values of observations in the state. Predicates of this kind are called *state predicates*. Given a state $v \in \Upsilon$, $c_\tau$ is true in $v$, namely $\tau$ is *effective* in $v$, iff $v \neq_{\mathcal{S}} \tau(v)$.

Multiple similar observations or transition rules may be indexed. Generally, observations and transition rules are denoted by $o_{i_1,\ldots,i_m}$ and $\tau_{j_1,\ldots,j_n}$, respectively, provided that $m, n \geq 0$ and we assume that there exist data types $D_k$ such that $k \in D_k$ $(k = i_1, \ldots, i_m, j_1, \ldots, j_n)$. For example, an integer array $a$ possessed by a process $p$ may be denoted by an observation $a_p$, and the increment of the $i$th element of the array may be denoted by a transition rule $inca_{p,i}$.

An execution starts from one initial state and goes on forever; in each step of execution some transition rule is selected nondeterministically and executed. Nondeterministic selection is constrained by the following fairness rule: every transition rule is selected infinitely often. Given an OTS, a set of infinite sequences of states is obtained from execution, constrained by the fairness rule, of the OTS. Such an infinite sequence of states is called an execution of the OTS. More specifically, an execution of an OTS $\mathcal{S}$ is an infinite sequence $s_0, s_1, \ldots$ of states satisfying:

- *Initiation*: For each $o \in \mathcal{O}$, $o(s_0)$ satisfies $\mathcal{I}$.
- *Consecution*: For each $i \in \{0, 1, \ldots\}$, $s_{i+1} =_{\mathcal{S}} \tau(s_i)$ for some $\tau \in \mathcal{T}$.
- *Fairness*: For each $\tau \in \mathcal{T}$, there exist an infinite number of indexes $i \in \{0, 1, \ldots\}$ such that $s_{i+1} =_{\mathcal{S}} \tau(s_i)$.

A state is called *reachable* with respect to $\mathcal{S}$ if it appears in an execution of the OTS.

**Verification** Important properties that an OTS may have are basically classified into two classes: *safety* and *liveness*

(or *progress*) properties. Since non-overcharge property is a safety property, we only describe safety properties and how to prove that an OTS has a safety property in this paper. Safety properties are defined as follows: a predicate $p : \Upsilon \rightarrow \{\text{true}, \text{false}\}$ is a safety property with respect to $\mathcal{S}$ iff $p$ is a state predicate and $p(v)$ holds for every reachable $v \in \Upsilon$.

If we prove that an OTS has a safety property $p$, the following induction is mainly used:

- Base case: For any state $v \in \Upsilon$ in which each observation $o \in \mathcal{O}$ satisfies $\mathcal{I}$, we show that $p(v)$ holds.
- Inductive step: Given any reachable state $v \in \Upsilon$ such that $p(v)$ holds, we show that, for any transition rule $\tau \in \mathcal{T}$, $p(\tau(v))$ also holds.

### 5.3. Observational Transition Systems in CafeOBJ

CafeOBJ[5, 9] is mainly based on two logical foundations: *initial* and *hidden* algebra. Initial algebra is used to specify abstract data types such as integers, and hidden algebra[12] to specify abstract machines. There are two kinds of sorts (corresponding to types in programming languages) in CafeOBJ. They are *visible* and *hidden* sorts. A visible sort represents an abstract data type, and a hidden sort the state space of an abstract machine. There are basically two kinds of operations to hidden sorts. They are *action* and *observation* operations. An action operation can change a state of an abstract machine. It takes a state of an abstract machine and zero or more data, and returns another (possibly the same) state of the abstract machine. Only observation operations can be used to observe the inside of an abstract machine. An observation operation takes a state of an abstract machine and zero or more data, and returns a value corresponding to the state. An action operation is basically specified with equations by describing how the value of each observation operation changes relatively based on the values of observation operations in a state after executing the action operation in the state.

Declarations of visible sorts are enclosed with `[` and `]`, and those of hidden ones with `*[` and `]*`. Declarations of observation and action operations start with `bop` or `bops`, and those of other operations with `op` or `ops`. After `bop` or `op` (or `bops` or `ops`), an operator is written (or more than one operator is written), followed by `:` and a sequence of sorts (i.e. sorts of the operators' arguments), and ended with `->` and one sort (i.e. sort of the operators' results). Definitions of equations start with `eq`, and those of conditional ones with `ceq`. After `eq`, two expressions, or terms connected by = are written, ended with a full stop. After `ceq`, two terms connected by = are written, followed by `if` and a term denoting a condition, and ended with a full stop.

The CafeOBJ system, an implementation of CafeOBJ, rewrites (reduces) a given term by regarding equations as left-to-right rewrite rules. This executability makes it possible to simulate described systems and to verify that they possess some desired properties.

Basic units of description by CafeOBJ are modules. Modules may have parameters, and generic lists, etc. can be described. Attributes such as associativity and commutativity may be given to operations, and sets and bags (multisets) can be conveniently declared. Each operation can be given local strategy as its attribute, and rewriting can be controlled to some extent. We can define the syntax of terms by declaring mix-fix operators. Thus, CafeOBJ has a lot of functionalities. We will mention other functionalities we do not here if we encounter them.

An OTS $\mathcal{S}$ is described in CafeOBJ.

The universal state space $\Upsilon$ is denoted by a hidden sort, say `Sys`, by declaring `*[Sys]*`.

An observation $o_{i_1,\ldots,i_m} \in \mathcal{O}$ is denoted by a CafeOBJ observation operation. We assume that data types $D_k$ $(k = i_1, \ldots, i_m)$ and $D$ are described in initial algebra and there exist visible sorts $S_k$ $(k = i_1, \ldots, i_m)$ and $S$ corresponding to the data types. The CafeOBJ observation operation denoting $o_{i_1,\ldots,i_m}$ is declared as follows:

```
bop o : Sys S_i1 ... S_im -> S
```

The initial condition $\mathcal{I}$, the value of each observation in any initial state, is described by declaring a constant (an operator without any arguments) denoting any initial state and specifying the value of each observation in the state with equations. First, the constant `init` denoting any initial state is declared as follows:

```
op init : -> Sys
```

Suppose that the initial value of $o_{i_1,\ldots,i_m}$ is $f(i_1, \ldots, i_m)$, this can be described in CafeOBJ as follows:

```
eq o(init,X_i1,...,X_im) = f(X_i1,...,X_im) .
```

where $\mathtt{X}_k$ $(k = i_1, \ldots, i_m)$ is a CafeOBJ variable which sort is $\mathtt{S}_k$, and `f(`$\mathtt{X}_{i_1}, \ldots, \mathtt{X}_{i_m}$`)` means a term (consisting of $\mathtt{X}_{i_1}, \ldots, \mathtt{X}_{i_m}$) corresponding to $f(i_1, \ldots, i_m)$.

A transition rule $\tau_{j_1,\ldots,j_n} \in \mathcal{T}$ is denoted by a CafeOBJ action operation. We assume that data types $D_k$ $(k = j_1, \ldots, j_n)$ are described in initial algebra and there exist visible sorts $S_k$ $(k = j_1, \ldots, j_n)$ corresponding to the data types. The CafeOBJ action operation denoting $\tau_{j_1,\ldots,j_n}$ is declared as follows:

```
bop a : Sys S_j1 ... S_jn -> Sys
```

If $\tau_{j_1,\ldots,j_n}$ is executed in a state in which it is effective, the value of $o_{i_1,\ldots,i_m}$ may be changed, which can be described in CafeOBJ generally as follows:

```
ceq o(a(S,Xj1,...,Xjn),Xi1,...,Xim)
    = e-a(S,Xj1,...,Xjn,Xi1,...,Xim)
      if c-a(S,Xj1,...,Xjn) .
```

where `e-a(S,`$X_{j_1},\ldots,X_{j_n},X_{i_1},\ldots,X_{i_m}$`)` means a term (consisting of $\mathtt{S}, \mathtt{X}_{j_1},\ldots,\mathtt{X}_{j_n},\mathtt{X}_{i_1},\ldots,\mathtt{X}_{i_m}$) corresponding to the value of $o_{i_1,\ldots,i_m}$ in the successor state, and `c-a(S,`$X_{j_1},\ldots,X_{j_n}$`)` means a term (consisting of $\mathtt{S}, \mathtt{X}_{j_1},\ldots,\mathtt{X}_{j_n}$) corresponding to $c_{\tau_{j_1,\ldots,j_n}}$.

If $\tau_{j_1,\ldots,j_n}$ is executed in a state in which it is not effective, the value of any observation is not changed. Therefore, all we have to do is to declare the following equation:

```
ceq a(S,Xj1,...,Xjn) = S if not c-a(S,Xj1,...,Xjn) .
```

If the value of $o_{i_1,\ldots,i_m}$ is not affected by executing $\tau_{j_1,\ldots,j_n}$ in any state (regardless of the truth value of $c_{\tau_{j_1,\ldots,j_n}}$), the following equation is declare:

```
eq o(a(S,Xj1,...,Xjn),Xi1,...,Xim) = o(S,Xi1,...,Xim) .
```

## 5.4. Modeling

Let us model as an OTS the AM3KP protocol. First data structures used in the protocol are defined in CafeOBJ.

**Data Structures Used to Model Abstract 3KP** Module BAN defines BAN's. The signature (in the context of algebraic specification) is as follows:

```
[Ban]
op _=_ : Ban Ban -> Bool {comm}
```

CafeOBJ provides built-in operator `_==_`, but it could be sometimes troublesome especially for verification. Therefore, for each data structure used to model the AM3KP protocol, we define operator `_=_` that checks if two values are equal. The operator is given operator attribute `comm` declaring that the operator is commutative. Necessary equations for defining operator `_=_` should be described. For visible sort `Ban`, we have the following equation:

```
eq (N:Ban = N) = true .
```

where `N` is a CafeOBJ variable which sort is `Ban`. Such a equation is called the basic equation for the operator. Every visible sort corresponding to data structure used to model the AM3KP protocol has this equation.

Module RCODE defines RESPCODE's. The signature is as follows:

```
[Rcode]
ops yes no : -> Rcode
op _=_     : Rcode Rcode -> Bool {comm}
```

Modules PKEY and SKEY define public and secret keys respectively. We have visible sorts `Pkey` and `Skey`, and the equality operator for each sort.

Module BUYER defines buyers. The signature is as follows:

```
[Buyer]
op b   : Ban -> Buyer
op ban : Buyer -> Ban
op pk  : Buyer -> Pkey
op sk  : Buyer -> Skey
op _=_ : Buyer Buyer -> Bool {comm}
```

Given a buyer, operators `ban`, `pk` and `sk` return the corresponding BAN, public key and secret key, respectively. On the other hand, given a BAN, operator `b` returns the corresponding buyer.

Module `Seller` defines sellers. The signature is as follows:

```
[Seller]
op pk  : Seller -> Pkey
op sk  : Seller -> Skey
op _=_ : Seller Seller -> Bool {comm}
```

Module ACQUIRER defines acquirers. The signature is as follows:

```
[Acquirer]
op pk  : Acquirer -> Pkey
op sk  : Acquirer -> Skey
op _=_ : Acquirer Acquirer -> Bool {comm}
```

Module RAND defines random numbers. The signature is as follows:

```
[Rand]
op nxt : Buyer Rand -> Rand
op gtr : Rand -> Buyer
op rnd : Rand -> Rand
op _=_ : Rand Rand -> Bool {comm}
```

Operator `nxt` denotes a perfect random number generator. The first argument indicates the buyer who has generated the random number, and the second argument can be regarded as something like a seed to generate the random number. Operators `gtr` and `rnd` are projection functions of a random number, returning the first and second arguments respectively.

Module HASHVALUE defines values returned by hash functions $\mathcal{H}$ and $\mathcal{H}_k$. Two visible sorts `Hban` and `Hcom`, and their equality operators are declared. `Hban` corresponds to $\mathcal{H}_k(\mathrm{R}_B, \mathrm{BAN})$, and `Hcom` to $\mathcal{H}(\mathrm{Common})$. The data constructors for the hash values are defoned after the definition of module COMMON because undefined symbols cannot be used in CafeOBJ, or forward reference is prohibited in CafeOBJ.

Module COMMON defines Common's. The signature is as follows:

```
[Common]
op com  : Seller Buyer Hban -> Common
op s    : Common -> Seller
op b    : Common -> Buyer
op hban : Common -> Hban
op _=_  : Common Common -> Bool {comm}
```

Operator `com` is the data structure, and the following three operators are the projection functions.

Module HCOM defines hash values returned by applying $\mathcal{H}$ to Common's. The data constructor `h` is declared. Module HBAN defines hash values returned by applying $\mathcal{H}_K$ to pairs of random numbers and BAN's. The data constructor

h and the projection function r returning the random number are declared.

Module CLEAR defines Clear's. The signature is as follows:

```
[Clear]
op cl   : Hcom -> Clear
op hcom : Clear -> Hcom
op _=_  : Clear Clear -> Bool {comm}
```

Operator cl is the data constructor, and the following operator is the projection function.

Module SLIP defines SLIP's. The signature is as follows:

```
[Slip]
op slp  : Hcom Ban Rand -> Slip
op hcom : Slip -> Hcom
op ban  : Slip -> Ban
op rand : Slip -> Rand
op _=_  : Slip Slip -> Bool {comm}
```

Operator slp is the data constructor, and the following three operators are the projection functions.

Module ESLP defines the ciphers obtained by encrypting SLIP's with a public key. The signature is as follows:

```
[Eslp]
op enc  : Pkey Slip -> Eslp
op pk   : Eslp -> Pkey
op slip : Eslp -> Slip
op _=_  : Eslp Eslp -> Bool {comm}
```

Operator enc is the data constructor, and the following two operators are the projection functions.

Module SIGA defines signatures (in the context of cryptography) generated by acquirers. The signature (in the context of algebraic specification) is as follows:

```
[Siga]
op sig : Skey Rcode Hcom -> Siga
op sk  : Siga -> Skey
op rc  : Siga -> Rcode
op hc  : Siga -> Hcom
op _=_ : Siga Siga -> Bool {comm}
```

Operator sig is the data constructor, and the following three operators are the projection functions.

Module SIGS defines signatures (in the context of cryptography) generated by sellers for sending Invoice. The signature (in the context of algebraic specification) is as follows:

```
[Sigs]
op sig : Skey Hcom -> Sigs
op sk  : Sigs -> Skey
op hc  : Sigs -> Hcom
op _=_ : Sigs Sigs -> Bool {comm}
```

Operator sig is the data constructor, and the following two operators are the projection functions.

Module SIGS2 defines signatures (in the context of cryptography) generated by sellers for sending Auth-Request. The signature (in the context of algebraic specification) is as follows:

```
[Sigs2]
op sig : Skey Hcom Eslp -> Sigs2
op sk  : Sigs2 -> Skey
op hc  : Sigs2 -> Hcom
op es  : Sigs2 -> Eslp
op _=_ : Sigs2 Sigs2 -> Bool {comm}
```

Operator sig is the data constructor, and the following three operators are the projection functions.

Module SIGB defines signatures (in the context of cryptography) generated by buyers. The signature (in the context of algebraic specification) is as follows:

```
[Sigb]
op sig : Skey Eslp Hcom -> Sigb
op sk  : Sigb -> Skey
op hc  : Sigb -> Hcom
op es  : Sigb -> Eslp
op _=_ : Sigb Sigb -> Bool {comm}
```

Operator sig is the data constructor, and the following three operators are the projection functions.

Module MSG defines messages exchanged by principals in abstract 3KP protocol. The main part of the signature is as follows:

```
[Msg]
op im : Buyer    Buyer    Seller   Hban          -> Msg
op vm : Seller   Seller   Buyer    Clear  Sigs -> Msg
op pm : Buyer    Buyer    Seller   Eslp   Sigb -> Msg
op qm : Seller   Seller   Acquirer Clear  Eslp
                                     Sigs2 Sigb -> Msg
op sm : Acquirer Acquirer Seller   Rcode  Siga -> Msg
```

Operators im, vm, pm, qm and sm are data constructors for Initiate, Invoice, Payment, Auth-Request and Auth-Response, respectively. The fist, second and third arguments of each constructor means the generator, the source and the destination of the corresponding message. The generator argument is a meta information that indicates who generates the message. You can understand what any other arguments of the constructors would be. Other than those operators, we have operators (im?, vm?, pm?, qm? and sm?) checking if a given message is a certain type of messages, and the projection functions. The projection functions are, for a given message, those (ic, vc, pc, qc, sc) returning the generator, those (is, vs, ps, qs, ss) returning the source, those (id, vd, pd, qd, sd) returning the destination, hban returning $ID_B$ if any, clear returning Clear if any, eslip returning EncSlip if any, rcode returning RCODE if any, siga returning $Sig_A$ if any, sigs returning $Sig_S$ if any, sigs2 returning $Sig2_S$ if any, and sigb returning $Sig_B$ if any. We also have the equality operator.

The network with which messages are exchanged is modeled as a bag (or multiset) of messages. Module EQTRIV is needed for defining module BAG. We have visible sort TRIV and the equality operator in the module.

Module BAG is a parameterized one with one parameter. The formal parameter is D :: EQTRIV. The signature is as follows:

```
[Elt.D < Bag]
op void  : -> Bag
op _,_   : Bag Bag -> Bag { assoc comm id: void }
op _\in_ : Elt.D Bag -> Bool
```

CafeOBJ is order-sorted, meaning that you can define partial ordering among sorts. By declaring Elt.D < Bag,

an element of bags can be regarded as a singleton bag. Operator `void` denotes the empty bag. Operator `_,_` is the data constructor of bags. Other than `comm`, the operator is given two more operator attributes `assoc` and `id: void`. The former means that the operator is associative, and the latter that `void` is the identity of bags. Given an element and a bag, operator `_\in_` checks if the bag includes the element.

Module `INTRUDER` defines an intruder. The intruder acts as each of a usual buyer, a usual seller and a usual acquirer. We have the following constants:

```
op ib : -> Buyer
op is : -> Seller
op ia : -> Acquirer
```

The intruder behaves basically following the Doleve-Yao general intruder model[10]. What the intrude can do other than as a usual principal in the protocol will be described.

The intruder can eavesdrop any message flowing in the network and glean information related to the protocol from the message. Given a snapshot of the network, what information the intruder has gleaned is represented by the collection data structure defined in parameterized module `COLLECTION` with one parameter. The formal parameter is `D::TRIV`, where `TRIV` is a built-in module in which one visible sort `Elt` is declared. The signature of module `COLLECTION` is as follows:

```
[Elt.D < Col]
op _\in_ : Elt.D Col -> Bool
```

Module `NETWORK` defines the network. The module imports several other modules as follows:

```
pr(INTRUDER)
pr(BAG(MSG)            *{sort Bag -> Network})
pr(COLLECTION(HCOM)    *{sort Col -> ColHcoms})
pr(COLLECTION(HBAN)    *{sort Col -> ColHbans})
pr(COLLECTION(BAN)     *{sort Col -> ColBans})
pr(COLLECTION(RAND)    *{sort Col -> ColRands})
pr(COLLECTION(ESLP)    *{sort Col -> ColEslps})
pr(COLLECTION(RCODE)   *{sort Col -> ColRcodes})
pr(COLLECTION(SIGA)    *{sort Col -> ColSigas})
pr(COLLECTION(SIGS)    *{sort Col -> ColSigss})
pr(COLLECTION(SIGS2)   *{sort Col -> ColSigs2s})
pr(COLLECTION(SIGB)    *{sort Col -> ColSigbs})
```

CafeOBJ command `pr` is basically used to import modules. Other than module `INTRUDER`, a module is first instantiated, a sort is renamed, and then the instantiated and renamed module is imported. For example, module `BAG` is instantiated with module `MSG`, creating a module defining bags of messages, the visible sort `Bag` in the created module is renamed to `Network`, and then it is imported. Visible sort `Network` is dedicated to the network with which messages are exchanged in the protocol. Visible sorts `Col`$X$ ($X$ = `Hcoms`, `Hbans`, `Bans`, `Rands`, `Eslps`, `Rcodes`, `Sigas`, `Sigss`, `Sigs2s`, `Sigbs`) correspond to quantities gleaned by the intruder from the network.

The signature of module `NETWORK` is as follows:

```
op hcoms  : Network -> ColHcoms
op hbans  : Network -> ColHbans
op bans   : Network -> ColBans
op rands  : Network -> ColRands
op eslps  : Network -> ColEslps
op rcodes : Network -> ColRcodes
op sigas  : Network -> ColSigas
op sigss  : Network -> ColSigss
op sigs2s : Network -> ColSigs2s
op sigbs  : Network -> ColSigbs
```

Given a network, the operators from top corresponds to hash values obtained by applying $\mathcal{H}$ to BAN's, hash values obtained by applying $\mathcal{H}_K$ to pairs of random numbers and a BAN's, BAN's, random numbers, EncSlip's, RESPCODE's, signatures generated by acquirers, signatures generated by sellers for sending Invoice, signatures generated by sellers for sending Auth-Request, and signatures by buyers, respectively, gleaned by the intruder from the network.

We are about to describe equations defining those operators because they are a key to modeling the protocol. While describing the equations, it is assumed that `NW`, `M`, `HC`, `HN`, `S`, `B`, `N`, `R`, `EP`, `C`, `GA`, `GS`, `GT` and `GB` are CafeOBJ variables which sorts are `Network`, `Msg`, `Hcom`, `Hban`, `Seller`, `Buyer`, `Ban`, `Rand`, `Eslp`, `Rcode`, `Siga`, `Sigs`, `Sigs2` and `Sigb`, respectively.

The equations for defining operator `hcoms` are as follows:

```
eq  HC \in hcoms(void)    = false .
ceq HC \in hcoms(M,NW)    = true
    if vm?(M) and HC = hcom(clear(M)) .
ceq HC \in hcoms(M,NW)    = true
    if pm?(M) and pk(ia) = pk(eslip(M)) and
       HC = hcom(slip(eslip(M))) .
ceq HC \in hcoms(M,NW)    = true
    if qm?(M) and HC = hcom(clear(M)) .
ceq HC \in hcoms(M,NW)    = true
    if qm?(M) and pk(ia) = pk(eslip(M)) and
       HC = hcom(slip(eslip(M))) .
ceq HC \in hcoms(M,NW)    = HC \in hcoms(NW)
    if not(vm?(M) and HC = hcom(clear(M))) and
       not(pm?(M) and pk(ia) = pk(eslip(M)) and
           HC = hcom(slip(eslip(M)))) and
       not(qm?(M) and HC = hcom(clear(M))) and
       not(qm?(M) and pk(ia) = pk(eslip(M)) and
           HC = hcom(slip(eslip(M)))) .
```

If the network does not have any message, the intruder has not gleaned any hashed Common's from the network. If the network includes Initiate or Auth-Request, the intruder can glean any hashed Common's in plain text. Besides, if the network includes Payment or Auth-Request, and EncSlip in the message happens to be encrypted by a public key (namely `pk(ia)`) known by the intruder, then the hashed Common occurring in the EncSlip can also be gleaned. Otherwise, the intruder cannot glean any hash values.

The equations for defining operator `hbans` are as follows:

```
eq  HN \in hbans(void) = false .
ceq HN \in hbans(M,NW) = true
    if im?(M) and HN = hban(M) .
ceq HN \in hbans(M,NW) = HN \in hbans(NW)
    if not(im?(M) and HN = hban(M)) .
```

The equations for defining operator bans are as follows:

```
eq  ban(ib) \in bans(void) = true .
ceq N \in bans(void) = false if not(N = ban(ib)) .
ceq N \in bans(M,NW) = true
    if pm?(M) and pk(ia) = pk(eslip(M)) and
       N = ban(slip(eslip(M))) .
ceq N \in bans(M,NW) = true
    if qm?(M) and pk(ia) = pk(eslip(M)) and
       N = ban(slip(eslip(M))) .
ceq N \in bans(M,NW) = N \in bans(NW)
    if not(pm?(M) and pk(ia) = pk(eslip(M)) and
           N = ban(slip(eslip(M)))) and
       not(qm?(M) and pk(ia) = pk(eslip(M)) and
           N = ban(slip(eslip(M)))) .
```

In any situation, the intruder knows his/her own BAN (namely ban(ib)) as a buyer.

The equations for defining operator rands are as follows:

```
eq R \in rands(void) = false .
ceq R \in rands(M,NW) = true
    if im?(M) and ib = gtr(r(hban(M))) and
       R = r(hban(M)) .
ceq R \in rands(M,NW) = true
    if pm?(M) and pk(ia) = pk(eslip(M)) and
       R = rand(slip(eslip(M))) .
ceq R \in rands(M,NW) = true
    if qm?(M) and pk(ia) = pk(eslip(M)) and
       R = rand(slip(eslip(M))) .
ceq R \in rands(M,NW) = R \in rands(NW)
    if not(im?(M) and ib = gtr(r(hban(M))) and
           R = r(hban(M))) and
       not(pm?(M) and pk(ia) = pk(eslip(M)) and
           R = rand(slip(eslip(M)))) and
       not(qm?(M) and pk(ia) = pk(eslip(M)) and
           R = rand(slip(eslip(M)))) .
```

The second equation states that the intruder knows random numbers generated by herself/himself.

The equations for defining operator eslps are as follows:

```
eq  EP \in eslps(void) = false .
ceq EP \in eslps(M,NW) = true
    if pm?(M) and EP = eslip(M) .
ceq EP \in eslps(M,NW) = true
    if qm?(M) and EP = eslip(M) .
ceq EP \in eslps(M,NW) = EP \in eslps(NW)
    if not(pm?(M) and EP = eslip(M)) and
       not(qm?(M) and EP = eslip(M)) .
```

The equations for defining operator rcodes are as follows:

```
eq  C \in rcodes(void) = false .
ceq C \in rcodes(M,NW) = true
    if sm?(M) and C = rcode(M) .
ceq C \in rcodes(M,NW) = C \in rcodes(NW)
    if not(sm?(M) and C = rcode(M)) .
```

The equations for defining operator sigas are as follows:

```
eq  GA \in sigas(void) = false .
ceq GA \in sigas(M,NW) = true
    if sm?(M) and GA = siga(M) .
ceq GA \in sigas(M,NW) = GA \in sigas(NW)
    if not(sm?(M) and GA = siga(M)) .
```

The equations for defining operator sigss are as follows:

```
eq  GS \in sigss(void) = false .
ceq GS \in sigss(M,NW) = true
    if vm?(M) and GS = sigs(M) .
ceq GS \in sigss(M,NW) = GS \in sigss(NW)
    if not(vm?(M) and GS = sigs(M)) .
```

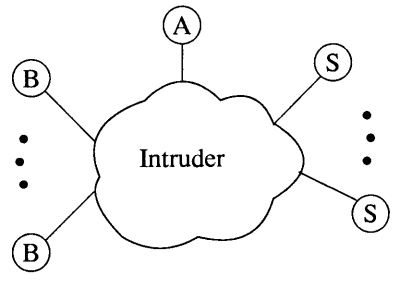


**Figure 6. The network as the intruder**

The equations for defining operator sigs2s are as follows:

```
eq  GT \in sigs2s(void) = false .
ceq GT \in sigs2s(M,NW) = true
    if qm?(M) and GT = sigs2(M) .
ceq GT \in sigs2s(M,NW) = GT \in sigs2s(NW)
    if not(qm?(M) and GT = sigs2(M)) .
```

The equations for defining operator sigbs are as follows:

```
eq  GB \in sigbs(void) = false .
ceq GB \in sigbs(M,NW) = true
    if pm?(M) and GB = sigb(M) .
ceq GB \in sigbs(M,NW) = true
    if qm?(M) and GB = sigb(M) .
ceq GB \in sigbs(M,NW) = GB \in sigbs(NW)
    if not(pm?(M) and GB = sigb(M)) and
       not(qm?(M) and GB = sigb(M)) .
```

As described now, the intruder can eavesdrop any message flowing in the network and, from it, glean any quantity except those cryptographically processed. The network is assumed to be completely under the intruder's control. Therefore, the network can be regarded as the intruder herself/himself as shown in Fig. 6.

All the intruder can do, other than those done by him/her as a usual principal, is to fake messages using values gleaned, as described now, from the network.

In the modeling, we assume that there is one and only legitimate acquirer, and every principal obtains his/her public key securely. Module LEGITIMATEACQUIRER defines the legitimate acquirer. We have two operators and one equation as follows:

```
op la    : -> Acquirer
op check : Ban -> Rcode
eq (la = ia) = false .
```

Constant la denotes the legitimate acquirer. Operator check corresponds to that the acquirer uses the authorization system to obtain on-line authorization of a payment. The equation declares that the legitimate acquirer is different from the intruder.

**Observational Transition System to Model AM3KP**

The universal state space $\Upsilon$ is denoted by hidden sort Protocol. To denote any initial state of the OTS, the following constant is declared:

```
op init : -> Protocol
```

We use two observations in the OTS. The corresponding CafeOBJ observation operators are declared as follows:

```
bop nw   : Protocol -> Network
bop rand : Protocol -> Rand
```

Operator `rand` denotes a perfect random number generator used in the protocol. As you can imagine, operator `nw` denotes the network.

We use, in the OTS, transition rules corresponding to passing messages, which are basically classified into two categories: one obeys the protocol, and the other fakes messages.

The CafeOBJ action operators corresponding to transition rules obeying the protocol are as follows:

```
bop sdim : Protocol Buyer  Seller          -> Protocol
bop sdvm : Protocol Seller Msg             -> Protocol
bop sdpm : Protocol Buyer  Rand Msg Msg    -> Protocol
bop sdqm : Protocol Seller Hban Msg Msg    -> Protocol
bop sdsm : Protocol Msg                    -> Protocol
```

Given a buyer and a seller, operator `sdim` corresponds to that the buyer sends Invoice to the seller.

Given a seller and a message, operator `sdvm` corresponds to that the seller sends Invoice to a buyer if the network includes the message, and the source and destination of the message are the buyer and the seller respectively. The conditoin does not include who has actually sent (created) the message because the receiver of a message cannot generally decide who has actually sent the message. Therefore who has actually sent the message may be the intruder. All the receiver can do is to look at the source field of the message and to expect that she/he is probably the sender.

Given a buyer, a random number and two messages `m1`, `m2`, operator `sdpm` corresponds to that the buyer sends Payment to a seller if the following holds. The network includes the two messages, `m1` is Initiate that the buyer has sent to the seller, $ID_B$ is computed using the random number, `m2` is Invoice which source is the seller and which destination is the buyer, and `m2` is the valid with respect to `m1` as described in Sect. 2. The buyer may remember that she/he has sent which messages to whom by storing them in her/his own memory. In the modeling, that a buyer has sent a message to a seller is decided by looking the creator and destination fields of the message. This does not mean that anyone can decide who has generated a message.

Given a seller, a hashed BAN and two messages `m1`, `m2`, operator `sdqm` corresponds to that the seller sends Auth-Request to the legitimate acquirer if the following holds. The network includes the two messages, `m1` is Invoice that the seller has generated using the hashed BAN and sent to a buyer, `m2` is Payment which source and destination are the buyer and the seller, `m2` is the valid with respect to `m1` as described in Sect. 2.

Given a message, operator `sdsm` corresponds to that the legitimate acquirer sends Auth-Response to a seller if the network includes the message which source and destination are the seller and the acquirer, and the message is valid as described in Sect. 2.

The transition rules faking messages are basically divided into five classes, each of which fakes each type of messages. The CafeOBJ action operators corresponding to transition rules faking Initiate are as follows:

```
bop fkim1 : Protocol Buyer  Seller Hash -> Protocol
bop fkim2 : Protocol Seller Ban    Rand -> Protocol
```

The CafeOBJ action operators corresponding to transition rules faking Invoice are as follows:

```
bop fkvm1 : Protocol Seller Buyer Hcom Sigs -> Protocol
bop fkvm2 : Protocol Seller Buyer Hban Sigs -> Protocol
bop fkvm3 : Protocol Seller Ban   Rand Sigs -> Protocol
bop fkvm4 : Protocol Seller Buyer Hcom      -> Protocol
bop fkvm5 : Protocol Seller Buyer Hban      -> Protocol
bop fkvm6 : Protocol Seller Ban   Rand      -> Protocol
```

The CafeOBJ action operators corresponding to transition rules faking Payment are as follows:

```
bop fkpm1 : Protocol Buyer  Seller
                                  Eslp Sigb -> Protocol
bop fkpm2 : Protocol Seller Ban
                                  Rand Hcom Sigb -> Protocol
bop fkpm3 : Protocol Seller Ban   Rand Sigb -> Protocol
bop fkpm4 : Protocol Buyer  Seller
                                  Eslp Hcom -> Protocol
bop fkpm5 : Protocol Buyer  Seller
                                  Eslp Hban -> Protocol
bop fkpm6 : Protocol Seller Eslp  Ban  Rand -> Protocol
bop fkpm7 : Protocol Seller Ban   Rand Hcom -> Protocol
bop fkpm8 : Protocol Seller Ban   Rand      -> Protocol
```

The CafeOBJ action operators corresponding to transition rules faking Auth-Request are as follows:

```
bop fkqm1  : Protocol Seller Hcom
                                Eslp Sigs2 Sigb -> Protocol
bop fkqm2  : Protocol Seller Buyer
                              Hban Eslp Sigs2 Sigb -> Protocol
bop fkqm3  : Protocol Seller Ban
                                Rand Sigs2 Sigb -> Protocol
bop fkqm4  : Protocol Seller Hcom Eslp Sigb -> Protocol
bop fkqm5  : Protocol Seller Buyer
                                 Hban Eslp Sigb -> Protocol
bop fkqm6  : Protocol Seller Ban  Rand Sigb -> Protocol
bop fkqm7  : Protocol Seller Hcom
                                     Eslp Sigs2 -> Protocol
bop fkqm8  : Protocol Seller Buyer
                                Hban Eslp Sigs2 -> Protocol
bop fkqm9  : Protocol Seller Ban Rand Sigs2 -> Protocol
bop fkqm10 : Protocol Seller Hcom Eslp      -> Protocol
bop fkqm11 : Protocol Seller Buyer
                                     Hban Eslp -> Protocol
bop fkqm12 : Protocol Seller Ban Rand       -> Protocol
```

The CafeOBJ action operators corresponding to transition rules faking Auth-Response are as follows:

```
bop fksm1 : Protocol Seller Rcode Siga     -> Protocol
bop fksm2 : Protocol Seller Rcode Hcom     -> Protocol
bop fksm3 : Protocol Seller Buyer Rcode
                                        Hban -> Protocol
bop fksm4 : Protocol Seller Rcode Ban Rand -> Protocol
```

Equations to define the action operators described now are shown in Fig. 7 through Fig. 10. In the equations, `P`, `B`, `S`, `M1`, `M2`, `R`, `N`, `HC`, `HN`, `EP`, `C`, `GA`, `GS`, `GT` and `GB` are CafeOBJ variables which sorts are `Protocol`, `Buyer`,

Seller, Msg, Msg, Rand, Ban, Hcom, Hban, Eslp, Rcode, Siga, Sigs, Sigs2 and Sigb, respectively.

The specification has 27 modules, and is approximately of 850 lines. The main module in which the OTS modeling abstract 3KP protocol is described is approximately of 300 lines.

## 6. Verification

That the legitimate acquirer receives valid Auth-Request implies that there exists the valid Auth-Request in the network. Besides, that there exists a message created by a principal implies that the principas has sent (generated) the message. Therefore agreement property with respect to the AM3KP protocol can be stated as follows:

**Claim 0.** *For any reachable* p : Protocol, *any* s1 s2 : Seller, *any* b1 : Buyer, *any* r1 : Rand,

```
not(s1 = is and b1 = ib)
and
qm(s2,s1,la,c1(h(com(s1,b1,h(r1,ban(b1))))),
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1))),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1)))))) \in nw(p)
implies
im(b1,b1,s1,h(r1,ban(b1))) \in nw(p)
and
vm(s1,s1,b1,c1(h(com(s1,b1,h(r1,ban(b1))))),
   sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))))) \in nw(p)
and
pm(b1,b1,s1,
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1)))))) \in nw(p)
and
qm(s1,s1,la,c1(h(com(s1,b1,h(r1,ban(b1))))),
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1))),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1)))))) \in nw(p)
```

The claim states that if, for any reachable state, there exists a valid Auth-Request message in the network, no matter who generates it, there always exist the corresponding Initiate and Payment messages generated by the involved buyer, and the corresponding Invoice and Auth-Request messages generated by the involved buyer.

To prove Claim 0, we need 17 more claims as lemmas. The claims are as follows:

**Claim 1.** *For any reachable* p : Protocol, *any* es1 : Cipher,

```
es1 \in eslps(nw(p)) implies pk(es1) = pk(la)
```

**Claim 2.** *For any reachable* p : Protocol, *any* n1 : Ban,

```
n1 \in bans(nw(p)) implies n1 = ban(ib)
```

**Claim 3.** *For any reachable* p : Protocol, *any* b1 : Buyer, *any* s1 : Seller, *any* r1 : Rand,

```
not(s1 = is)
and
sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))),
    enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                   ban(b1),r1))) \in sigs2s(nw(p))
implies
qm(s1,s1,la,c1(h(com(s1,b1,h(r1,ban(b1))))),
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1))),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1)))))) \in nw(p)
```

**Claim 4.** *For any reachable* p : Protocol, *any* s1 : Seller, *any* c11 : Clear, *any* es1 : Eslp, *any* st1 : Sigs2, *any* sb1 : Sigb,

```
qm(s1,is,la,c11,es1,st1,sb1) \in nw(p) implies s1 = is
```

**Claim 5.** *For any reachable* p : Protocol, *any* b1 : Buyer, *any* s1 s2 : Seller, *any* r1 : Rand,

```
qm(s2,s1,la,c1(h(com(s1,b1,h(r1,ban(b1))))),
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1))),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1)))))) \in nw(p)
implies
qm(s1,s1,la,c1(h(com(s1,b1,h(r1,ban(b1))))),
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1))),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1)))))) \in nw(p)
```

**Claim 6.** *For any reachable* p : Protocol, *any* b1 : Buyer, *any* s1 : Seller, *any* r1 : Rand,

```
not(s1 = is)
and
qm(s1,s1,la,c1(h(com(s1,b1,h(r1,ban(b1))))),
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1))),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1)))))) \in nw(p)
implies
vm(s1,s1,b1,c1(h(com(s1,b1,h(r1,ban(b1))))),
   sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))))) \in nw(p)
```

**Claim 7.** *For any reachable* p : Protocol, *any* s1 : Seller, *any* b1 : Buyer, *any* cl1 : Clear, *any* ss1 : Sigs,

```
vm(s1,is,b1,cl1,ss1) \in nw(p) implies s1 = is
```

**Claim 8.** *For any reachable* p : Protocol, *any* b1 : Buyer, *any* r1 : Rand,

```
not(b1 = ib)
and
enc(pk(la),slp(h(com(is,b1,h(r1,ban(b1)))),
               ban(b1),r1)) \in eslps(nw(p))
implies
vm(is,is,b1,cl(h(com(is,b1,h(r1,ban(b1))))),
   sig(sk(is),h(com(is,b1,h(r1,ban(b1))))) \in nw(p)
```

**Claim 9.** *For any reachable* p : Protocol, *any* b1 : Buyer, *any* s1 s2 : Seller, *any* r1 : Rand,

```
not(s1 = is and b1 = ib)
and
qm(s2,s1,la,cl(h(com(s1,b1,h(r1,ban(b1))))),
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1))),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1))))))) \in nw(p)
implies
vm(s1,s1,b1,cl(h(com(s1,b1,h(r1,ban(b1))))),
   sig(sk(s1),h(com(s1,b1,h(r1,ban(b1))))))) \in nw(p)
```

**Claim 10.** *For any reachable* p : Protocol, *any* b1 : Buyer, *any* s1 : Seller, *any* r1 : Rand,

```
not(b1 = ib)
and
enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
               ban(b1),r1)) \in eslps(nw(p))
implies
pm(b1,b1,s1,
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1))))))) \in nw(p)
```

**Claim 11.** *For any reachable* p : Protocol, *any* s1 : Seller, *any* b1 : Buyer, *any* es1 : Eslp, *any* sb1 : Sigb,

```
pm(b1,ib,s1,es1,sb1) \in nw(p) implies b1 = ib
```

**Claim 12.** *For any reachable* p : Protocol, *any* s1 : Seller, *any* r1 : Rand,

```
not(s1 = is)
and
sig(sk(s1),h(com(s1,ib,h(r1,ban(ib)))),
    enc(pk(la),slp(h(com(s1,ib,h(r1,ban(ib)))),
                   ban(ib),r1))) \in sigs2s(nw(p))
implies
pm(ib,ib,s1,
   enc(pk(la),slp(h(com(s1,ib,h(r1,ban(ib)))),
                  ban(ib),r1)),
   sig(sk(ib),
       enc(pk(la),slp(h(com(s1,ib,h(r1,ban(ib)))),
                      ban(ib),r1)),
       h(com(s1,ib,h(r1,ban(ib))))))) \in nw(p)
```

**Claim 13.** *For any reachable* p : Protocol, *any* b1 : Buyer, *any* s1 s2 : Seller, *any* r1 : Rand,

```
not(s1 = is and b1 = ib)
and
qm(s2,s1,la,cl(h(com(s1,b1,h(r1,ban(b1))))),
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1))),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1))))))) \in nw(p)
implies
pm(b1,b1,s1,
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1))))))) \in nw(p)
```

**Claim 14.** *For any reachable* p : Protocol, *any* b1 : Buyer, *any* s1 : Seller, *any* r1 : Rand,

```
not(b1 = ib)
and
pm(b1,b1,s1,
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1))))))) \in nw(p)
implies
im(b1,b1,s1,h(r1,ban(b1))) \in nw(p)
```

**Claim 15.** *For any reachable* p : Protocol, *any* s1 : Seller, *any* b1 : Buyer, *any* hn1 : Hban,

```
im(b1,ib,s1,hn1) \in nw(p) implies b1 = ib
```

**Claim 16.** *For any reachable* p : Protocol, *any* s1 : Seller, *any* r1 : Rand,

```
not(s1 = is)
and
sig(sk(s1),h(com(s1,ib,h(r1,ban(ib))))) \in sigss(nw(p))
implies
im(ib,ib,s1,h(r1,ban(ib))) \in nw(p)
```

**Claim 17.** *For any reachable* p : Protocol, *any* b1 : Buyer, *any* s1 s2 : Seller, *any* r1 : Rand,

```
not(s1 = is and b1 = ib)
and
qm(s2,s1,la,cl(h(com(s1,b1,h(r1,ban(b1))))),
   enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                  ban(b1),r1)),
   sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1))),
   sig(sk(b1),
       enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                      ban(b1),r1)),
       h(com(s1,b1,h(r1,ban(b1))))))) \in nw(p)
implies
im(b1,b1,s1,h(r1,ban(b1))) \in nw(p)
```

Claim 0 ,Claim 5, Claim 9, Claim 13 and Claim 17 have been proved only by case analysis, and the remaining by induction described in Sect. 5.2. All the proofs have been

done by writing proof scores in CafeOBJ and having the CafeOBJ system execute the proof scores.

In this paper, we describe parts of the proofs for Claim 8 and Claim 9 that are proved by induction and only by case analysis, respectively.

**Proof of Claim 8** We first write a module in which the predicate to be proved is declared. In the module called `PRED8`, two constants (`b1`, `r1`) denoting any intended sorts (you can imagine) are declared. Operator `p8` denoting the predicate is declared as follows:

```
op p8 : Protocol Buyer Rand -> Bool
```

We also have the equation defining the operator as follows:

```
eq p8(P,B1,R1)
   = not(B1 = ib)
     and
     enc(pk(la),slp(h(com(is,B1,h(R1,ban(B1)))),
                    ban(B1),R1)) \in eslps(nw(P))
     implies
     vm(is,is,B1,cl(h(com(is,B1,h(R1,ban(B1))))),
        sig(sk(is),h(com(is,B1,h(R1,ban(B1))))))
      \in nw(P) .
```

where `P`, `S10`, `T10`, `V10`, `N10` and `R10` are CafeOBJ variables which sorts may be imagined.

**Base Case** In any initial state `init`, to show that the predicate holds, the following proof score is described and executed by the CafeOBJ system:

```
open PRED8
red p8(init,b1,r1) .
close
```

**Inductive Step** The predicate to be proved in each inductive step is defined in module `ISTEP8`, in which two constants ($p$, $p'$) are declared, where $p$ and $p'$ denote any reachable state and the successor state after executing a transition rule in the state. The predicate is declared as follows:

```
op istep8 : Buyer Rand -> Bool
```

The equation defining the predicate is as follows:

```
eq istep8(B1,R1) = p8(p,B1,R1) implies p8(p',B1,R1) .
```

All we have to do is basically to show `istep8(b1,r1)` for every transition rule. In this paper, we describe proof scores showing that any transition rule denoted by action operator `fkpm2` preserves predicate `p5`. We first consider two cases such that one corresponds to any state in which the transition rule is effective and the other to any state in which the transition rule is not. Since the predicate is neither true nor false in the former case, the case is also split into two parts such that one corresponds to any state in which `b1` equals `ib`, and the other to any state in which `b1` does not. Furthermore, for the latter case, we make use of Claim 2. After all, we have three proof scores to show that the transition rule preserves the predicate. The three proof scores are as follows:

```
open ISTEP8
-- arbitrary chosen objects
op s10 : -> Seller .
op n10 : -> Ban .
op r10 : -> Rand .
op hc10 : -> Hcom .
op sb10 : -> Sigb .
-- assumptions
eq n10 \in bans(nw(p)) = true .
eq r10 \in rands(nw(p)) = true .
eq hc10 \in hcoms(nw(p)) = true .
eq sb10 \in sigbs(nw(p)) = true .
--
eq b1 = ib .
-- successor state
eq p' = fkpm2(p,s10,n10,r10,hc10,sb10) .
-- check if the predicate is also true in p'.
red istep8(b1,r1) .
close

open ISTEP8
-- arbitrary chosen objects
op s10 : -> Seller .
op n10 : -> Ban .
op r10 : -> Rand .
op hc10 : -> Hcom .
op sb10 : -> Sigb .
-- assumptions
-- eq n10 \in bans(nw(p)) = true .
eq ban(ib) \in bans(nw(p)) = true .
eq r10 \in rands(nw(p)) = true .
eq hc10 \in hcoms(nw(p)) = true .
eq sb10 \in sigbs(nw(p)) = true .
--
eq (b1 = ib) = false .
-- facts, etc.
-- from Claim 2 (n10 \in bans(nw(p)))
eq n10 = ban(ib) .
-- successor state
eq p' = fkpm2(p,s10,n10,r10,hc10,sb10) .
-- check if the predicate is also true in p'.
red istep8(b1,r1) .
close

open ISTEP8
-- arbitrary chosen objects
op s10 : -> Seller .
op n10 : -> Ban .
op r10 : -> Rand .
op hc10 : -> Hcom .
op sb10 : -> Sigb .
-- assumptions
eq (n10 \in bans(nw(p)) and r10 \in rands(nw(p)) and
    hc10 \in hcoms(nw(p)) and sb10 \in sigbs(nw(p)))
   = false .
-- successor state
eq p' = fkpm2(p,s10,n10,r10,hc10,sb10) .
-- check if the predicate is also true in p'.
red istep8(b1,r1) .
close
```

The number of cases to be considered in all the inductive steps is 102.

**Proof of Claim 9** Like the proof of Claim 8, module `PRED9` in which four constants (`b1`, `s1`, `s2`, `r1`) and operator `p9` are declared is first written. Operator `p9` is declared as follows:

```
op p9 : Protocol Buyer Seller Seller Rand -> Bool
```

We also have the equation defining the operator as follows:

```
eq p9(P,B1,S1,S2,R1)
   = not(S1 = is and B1 = ib)
     and
     qm(S2,S1,la,cl(h(com(S1,B1,h(R1,ban(B1))))),
        enc(pk(la),slp(h(com(S1,B1,h(R1,ban(B1)))),
                       ban(B1),R1)),
        sig(sk(S1),h(com(S1,B1,h(R1,ban(B1)))),
            enc(pk(la),slp(h(com(S1,B1,h(R1,ban(B1)))),
                           ban(B1),R1)),
        sig(sk(B1),
            enc(pk(la),slp(h(com(S1,B1,h(R1,ban(B1)))),
                           ban(B1),R1)),
            h(com(S1,B1,h(R1,ban(B1))))) \in nw(P)
     implies
     vm(S1,S1,B1,cl(h(com(S1,B1,h(R1,ban(B1))))),
        sig(sk(S1),h(com(S1,B1,h(R1,ban(B1))))))
      \in nw(P) .
```

where P, B1, S1 and R1 are CafeOBJ variables which sorts may be imagined.

Then we show that the predicate is always true in any reachable state p. We first consider two cases such that one corresponds to any state in which s1 equals is, and the other to any state in which s1 does not.

In this paper, we show the proof scores for the former case. The case is also split into two more cases such that one corresponds to any state in which b1 equals is, and the other to any state in which b1 does not. Furthermore, the latter case is split into two more cases such that one corresponds to any state in which there exists the Auth-Request message, in the network, corresponding to that occurring in the premise of the predicate, and the other to any state in which there does not. After all, we have three cases to be considered. For the second case, we make use of Claim 8. The three proof scores are as follows:

```
open PRED9
-- arbitrary chosen objects
op p : -> Protocol .
-- assumptions
eq s1 = is .
--
eq b1 = ib .
-- check if the predicate is also true in p.
red p9(p,b1,s1,s2,r1) .
close

open PRED9
-- arbitrary chosen objects
op p : -> Protocol .
op m10 : -> Msg .
op nw10 : -> Network .
-- assumptions
eq s1 = is .
--
eq (b1 = ib) = false .
eq m10
   = qm(s2,s1,la,cl(h(com(s1,b1,h(r1,ban(b1))))),
        enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                       ban(b1),r1)),
        sig(sk(s1),h(com(s1,b1,h(r1,ban(b1)))),
            enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                           ban(b1),r1))),
        sig(sk(b1),
            enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                           ban(b1),r1)),
            h(com(s1,b1,h(r1,ban(b1))))) .
eq nw(p) = m10 , nw10 .
-- facts, etc.
-- from Claim 8
eq vm(is,is,b1,cl(h(com(is,b1,h(r1,ban(b1))))),
      sig(sk(is),h(com(is,b1,h(r1,ban(b1))))))
    \in nw10 = true .
-- check if the predicate is also true in p.
red p9(p,b1,s1,s2,r1) .
close

open PRED9
-- arbitrary chosen objects
op p : -> Protocol .
-- assumptions
eq is = s1 .
--
eq (b1 = ib) = false .
eq qm(s2,s1,la,cl(h(com(s1,b1,h(r1,ban(b1))))),
      enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                     ban(b1),r1)),
      sig(sk(s1),h(com(s1,b1,h(r1,ban(b1))))),
          enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                         ban(b1),r1))),
      sig(sk(b1)),
          enc(pk(la),slp(h(com(s1,b1,h(r1,ban(b1)))),
                         ban(b1),r1)),
          h(com(s1,b1,h(r1,ban(b1))))))
    \in nw(p) = false .
-- check if the predicate is also true in p.
red p9(p,b1,s1,s2,r1) .
close
```

The size of all the proof scores is approximately of 22,000 lines. It took about 4 minutes to have the CafeOBJ system load the specification and execute the proof scores on a laptop with 850MHz Pentium III processor and 512MB memory.

The verification that the modified 2KP protocol (actually the AM2KP protocol) possess agreement property is very similar to that for the AM3KP protocol. Basically by deleting parts related to $\mathrm{Sig}_B$ from the CafeOBJ document and proof scores for the AM3KP protocol, we can get the proof for the AM2KP protocol.

## 7. Related Work

Recently applying formal methods to security protocols is one of the hottest research topics. They are most likely classified into two approaches: one is to find security flaws lurked in security protocols with model checking technology[8, 16], and the other to prove that security protocols have desired properties with theorem proving technology[28, 29, 30]. They are not competing, but complementary to each other. Our approach is classified into the latter.

Among security protocols, those to which formal methods have been often applied so far are authentication protocols such as the Kerberos authentication system[2, 1] because they are one of the most basic and important security protocols. It does not seem that there are many case studies to apply formal methods to payment protocols. To the best of our knowledge, we are the first to fromaly analyze the $i$KP protocols.

Our modeling is similar to the inductive method[28] and the CSP approach[29, 30].

Paulson[28] models a system in which an arbitrary number of principals including the intruder take part in an authentication protocol by inductively defining traces from a set of rules that correspond to the possible actions of the principals including the intruder. Security properties can be stated as predicates over the traces. You can inductively prove that a certain property holds of all possible traces for a certain protocol. The proof can be supported by the the theorem prover Isabelle/HOL[27].

Schneider[29, 30] models each principal participating in an authentication protocol as a CSP process. Proving a certain property corresponds to finding a rank function (which maps messages to some domain of values) satisfying certain conditions. The proof can be supported by the theorem prover PVS[31, 26].

In both approaches, the network is modeled by regarding it as the intruder, namely the intruder can glean anything flowing in the network except for those encrypted by a key that the intruder does not know and those hashed. We also model the network as this.

Observational transition systems are heavily affected by UNITY[6]. By adopting the notion of behavioral specification by hidden algebra[12] and the style of defining fair transition systems[18, 19], they are reformulated. The concept *effectiveness* is similar to *enabledness* used in description of transition systems in temporal logic such as TLA[14] or in a precondition-effect style such as I/O automata[17].

## 8. Concluding Remarks

We first tried to verify that the 3KP protocol has agreement property by assuming that AUTHPRICE and DESC might happen to be the same as another AUTHPRICE and DESC. We found out a counter example in the process. It took about a week to find it. After that, we retried the verification by assuming that $\mathrm{SALT}_B$ is never transmitted in clear, namely that $\mathrm{SALT}_B$ is encrypted with the seller's public key and sent to a seller. Unfortunately we found the counter example shown in Fig. 3 (2). Even if $\mathrm{SALT}_B$ is never transmitted in clear, the counter example can be occurred. Since our approach presented in this paper does not directly help us find counter examples, it took several weeks to reach the modified $i$KP protocols and the verification descrined in this paper. If we had used model checking techniques[7] to confirm that a system in which a finite number of principals participate in the 3KP protocol or its variant has no counter example with respect to agreement property, we would have reached the modified $i$KP protocols and the verification much earlier.

Although the designers of the $i$KP protocols notice that the 1KP protocol does not have agreement protocol, they write, on their papers[4, 3], that if $h_1$ retrieved from Clear and $h_2$ retrieved from decrypted EncSlip match when acquirer receives Auth-Request, it ensures that buyer and seller agree on the order information even for 1KP. But this is not true as shown by the counter example in Fig. 3 (1). Even so, we do not think that the designers and the reviewers of the papers were somehow careless. We think that the $i$KP protocols are rather well designed. We believe that security protocols are that sensitive and therefore should be formally verified.

In this paper, we have described the proof that the AM3KP protocol has agreement property, which we believe implies that the modified 3KP protocol also has the property. Although the relation between the modified 3KP protocol and the AM3KP protocol is very straightforward, to prove it more formally, we should show that there exists a relation between the 3KP protocol and the AM3KP protocol such as a simulation relation between I/O automata[17]. We could do this with CafeOBJ.

Other than agreement property, there are several properties that electronic payment protocols should have. Replay attack protection is one of such interesting properties. Since a variant of an OTS, called a TOTS[23], describes timing and verifies timing properties, we could prove that the (modified) $i$KP protocols do have replay attack protection.

## References


[1] G. Bella and L. C. Paulson. Using isabelle to prove properties of the kerberos authentication system. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[2] G. Bella and E. Riccobene. Formal analysis of the kerberos authentication system. *Journal of Universal Computer Science*, 3(12):1337–1381, 1997.

[3] M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, E. V. Herreweghen, and M. Waidner. Design, implementation and deployment of the *i*KP secure electronic payment system. *IEEE Journal of Selected Areas in Communications*, 18(4):611–627, 2000.

[4] M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. *i*KP – a family of secure electronic payment protocols. In *Proc. of First USENIX Workshop on Electronic Commerce*, pages 89–106, 1995.

[5] CafeOBJ web page. http://www.ldl.jaist.ac.jp/cafeobj/.

[6] K. M. Chandy and J. Misra. *Parallel program design: a foundation*. Addison-Wesley, Reading, MA, 1988.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, MA, 1999.

[8] G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in Maude. In *Formal Methods and Security Protocols Workshop (http://www.cs.bell-labs.com/who/nch/fmsp/)*, 1998.

[9] R. Diaconescu and K. Futatsugi. *CafeOBJ report*. AMAST Series in Computing, 6. World Scientific, Singapore, 1998.

[10] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Trans. Inform. Theory*, IT-29:198–208, 1983.

[11] J. Goguen and G. Malcolm. *Algebraic Semantics of Imperative Programs*. The MIT Press, Cambridge, MA, 1996.
[12] J. Goguen and G. Malcolm. A hidden agenda. *Theor. Comput. Sci.*, 245:55–101, 2000.
[13] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J. P. Jouannaud. Introducing OBJ. In J. Goguen and G. Malcolm, editors, *Software Engineering with OBJ: algebraic specification in action*. Kluwer Academic Publishers, 2000.
[14] L. Lamport. The temporal logic of actions. *ACM TOPLAS*, 16(3):872–923, 1994.
[15] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56:131–133, 1995.
[16] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS '96*, LNCS 1055, pages 147–166. Springer, 1996.
[17] N. A. Lynch. *Distributed algorithms*. Morgan-Kaufmann, San Francisco, CA, 1996.
[18] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: specification*. Springer-Verlag, NY, 1991.
[19] Z. Manna and A. Pnueli. *Temporal verification of reactive systems: safety*. Springer-Verlag, NY, 1995.
[20] MasterCard/Visa. SET secure electronic transactions protocol. Book One: Business Specifications, Book Two: Technical Specification, Book Three: Formal Protocol Definition (http://www.setco.org/set_specifications.html), May 1997.
[21] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Comm. ACM*, 21(12):993–999, 1978.
[22] K. Ogata and K. Futatsugi. Formally modeling and verifying Ricart&Agrawala distributed mutual exclusion algorithm. In *APAQS '01*, pages 357–366. IEEE CS Press, 2001.
[23] K. Ogata and K. Futatsugi. Modeling and verification of distributed real-time systems based on CafeOBJ. In *ASE '01*, pages 185–192. IEEE CS Press, 2001.
[24] K. Ogata and K. Futatsugi. Formal analysis of Suzuki&Kasami distributed mutual exclusion algorithm. In *FMOODS '02*, pages 181–195. Kluwer Academic Publishers, 2002.
[25] K. Ogata and K. Futatsugi. Rewriting-based verification of authentication protocols. In *WRLA '02*, volume 71 of *ENTCS*. Elsevier Science Publishers, 2002.
[26] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Fromal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Trans. Softw. Eng.*, 21:107–125, 1995.
[27] L. C. Paulson. *Isabelle: A generic theorem prover*. LNCS 828. Springer, 1994.
[28] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Comput. Security*, 6:85–128, 1998.
[29] S. Schneider. Verifying authentication protocols in CSP. *IEEE Trans. Softw. Eng.*, 24(9):741–758, 1998.
[30] S. Schneider. Verifying authentication protocol implementations. In *FMOODS '02*, pages 5–24. Kluwer Academic Publishers, 2002.
[31] N. Shanker, S. Owre, and J. M. Rushby. The PVS proof checker: A reference manual. SRI International, CSL Technical Report, 1993.

```
-- for any initial state
eq nw(init) = void .
-- for action sdim
eq nw(sdim(P,B,S)) = im(B,B,S,h(nxt(B,rand(P)),ban(B))) , nw(P) .
eq rand(sdim(P,B,S)) = nxt(B,rand(P)) .
-- for action sdvm
ceq nw(sdvm(P,S,M1))
    = vm(S,S,is(M1),cl(h(com(S,is(M1),hban(M1)))),sig(sk(S),h(com(S,is(M1),hban(M1))))) , nw(P)
    if M1 \in nw(P) and im?(M1) and S = id(M1) .
ceq nw(sdvm(P,S,M1)) = nw(P)
    if not(M1 \in nw(P) and im?(M1) and S = id(M1)) .
eq rand(sdvm(P,S,M1)) = rand(P) .
-- for action sdpm
ceq nw(sdpm(P,B,R,M1,M2))
    = pm(B,B,vs(M2),enc(pk(la),slp(hcom(clear(M2)),ban(B),R)),
                    sig(sk(B),enc(pk(la),slp(hcom(clear(M2)),ban(B),R)),hcom(clear(M2)))) , nw(P)
    if M1 \in nw(P) and im?(M1) and B = ic(M1) and B = is(M1) and hban(M1) = h(R,ban(B)) and
       M2 \in nw(P) and vm?(M2) and B = vd(M2) and id(M1) = vs(M2) and
       sig(sk(vs(M2)),hcom(clear(M2))) = sigs(M2) and hcom(clear(M2)) = h(com(id(M1),B,hban(M1))) .
ceq nw(sdpm(P,B,R,M1,M2)) = nw(P)
    if not(M1 \in nw(P) and im?(M1) and B = ic(M1) and B = is(M1) and hban(M1) = h(R,ban(B)) and
           M2 \in nw(P) and vm?(M2) and B = vd(M2) and id(M1) = vs(M2) and
           sig(sk(vs(M2)),hcom(clear(M2))) = sigs(M2) and hcom(clear(M2)) = h(com(id(M1),B,hban(M1)))) .
eq rand(sdpm(P,B,R,M1,M2)) = rand(P) .
-- for action sdqm
ceq nw(sdqm(P,S,HN,M1,M2))
    = qm(S,S,la,clear(M1),eslip(M2),sig(sk(S),h(com(S,ps(M2),HN)),eslip(M2)),sigb(M2)) , nw(P)
    if M1 \in nw(P) and vm?(M1) and S = vc(M1) and S = vs(M1) and
       hcom(clear(M1)) = h(com(S,ps(M2),HN)) and sigs(M1) = sig(sk(S),hcom(clear(M1))) and
       M2 \in nw(P) and pm?(M2) and vd(M1) = ps(M2) and S = pd(M2) and
       sigb(M2) = sig(sk(ps(M2)),eslip(M2),hcom(clear(M1))) .
ceq nw(sdqm(P,S,HN,M1,M2)) = nw(P)
    if not(M1 \in nw(P) and vm?(M1) and S = vc(M1) and S = vs(M1) and
           hcom(clear(M1)) = h(com(S,ps(M2),HN)) and sigs(M1) = sig(sk(S),hcom(clear(M1))) and
           M2 \in nw(P) and pm?(M2) and vd(M1) = ps(M2) and S = pd(M2) and
           sigb(M2) = sig(sk(ps(M2)),eslip(M2),hcom(clear(M1)))) .
eq rand(sdqm(P,S,HN,M1,M2)) = rand(P) .
-- for action sdsm
ceq nw(sdsm(P,M1))
    = sm(la,la,qs(M1),check(ban(slip(eslip(M1)))),
                      sig(sk(la),check(ban(slip(eslip(M1)))),hcom(clear(M1)))) , nw(P)
    if M1 \in nw(P) and qm?(M1) and la = qd(M1) and pk(la) = pk(eslip(M1)) and
       sig(sk(qs(M1)),hcom(clear(M1)),eslip(M1)) = sigs2(M1) and
       sig(sk(b(ban(slip(eslip(M1))))),eslip(M1),hcom(clear(M1))) = sigb(M1) and
       hcom(clear(M1)) = hcom(slip(eslip(M1))) and
       hcom(clear(M1)) = h(com(qs(M1),b(ban(slip(eslip(M1))))),
                           h(rand(slip(eslip(M1))),ban(slip(eslip(M1))))) .
ceq nw(sdsm(P,M1)) = nw(P)
    if not(M1 \in nw(P) and qm?(M1) and la = qd(M1) and pk(la) = pk(eslip(M1)) and
           sig(sk(qs(M1)),hcom(clear(M1)),eslip(M1)) = sigs2(M1) and
           sig(sk(b(ban(slip(eslip(M1))))),eslip(M1),hcom(clear(M1))) = sigb(M1) and
           hcom(clear(M1)) = hcom(slip(eslip(M1))) and
           hcom(clear(M1)) = h(com(qs(M1),b(ban(slip(eslip(M1))))),
                               h(rand(slip(eslip(M1))),ban(slip(eslip(M1)))))) .
eq rand(sdsm(P,M1)) = rand(P) .
```

**Figure 7. Equations to define transition rules corresponding to passing messages obeying the protocol**

```
-- for action fkim1
ceq nw(fkim1(P,B,S,HN)) = im(ib,B,S,HN) , nw(P) if HN \in hbans(nw(P)) .
ceq nw(fkim1(P,B,S,HN)) = nw(P) if not(HN \in hbans(nw(P))) .
eq rand(fkim1(P,B,S,HN)) = rand(P) .
-- for action fkim2
ceq nw(fkim2(P,S,N,R)) = im(ib,b(N),S,h(R,N)) , nw(P) if N \in bans(nw(P)) and R \in rands(nw(P)) .
ceq nw(fkim2(P,S,N,R)) = nw(P) if not(N \in bans(nw(P)) and R \in rands(nw(P))) .
eq rand(fkim2(P,S,N,R)) = rand(P) .
-- for action fkvm1
ceq nw(fkvm1(P,S,B,HC,GS)) = vm(is,S,B,cl(HC),GS) , nw(P)
    if HC \in hcoms(nw(P)) and GS \in sigss(nw(P)) .
ceq nw(fkvm1(P,S,B,HC,GS)) = nw(P)
    if not(HC \in hcoms(nw(P)) and GS \in sigss(nw(P))) .
eq rand(fkvm1(P,S,B,HC,GS)) = rand(P) .
-- for action fkvm2
ceq nw(fkvm2(P,S,B,HN,GS)) = vm(is,S,B,cl(h(com(S,B,HN))),GS) , nw(P)
    if HN \in hbans(nw(P)) and GS \in sigss(nw(P)) .
ceq nw(fkvm2(P,S,B,HN,GS)) = nw(P)
    if not(HN \in hbans(nw(P)) and GS \in sigss(nw(P))) .
eq rand(fkvm2(P,S,B,HN,GS)) = rand(P) .
-- for action fkvm3
ceq nw(fkvm3(P,S,N,R,GS)) = vm(is,S,b(N),cl(h(com(S,b(N),h(R,N)))),GS) , nw(P)
    if N \in bans(nw(P)) and R \in rands(nw(P)) and GS \in sigss(nw(P)) .
ceq nw(fkvm3(P,S,N,R,GS)) = nw(P)
    if not(N \in bans(nw(P)) and R \in rands(nw(P)) and GS \in sigss(nw(P))) .
eq rand(fkvm3(P,S,N,R,GS)) = rand(P) .
-- for action fkvm4
ceq nw(fkvm4(P,S,B,HC)) = vm(is,S,B,cl(HC),sig(sk(is),HC)) , nw(P)
    if HC \in hcoms(nw(P)) .
ceq nw(fkvm4(P,S,B,HC)) = nw(P)
    if not(HC \in hcoms(nw(P))) .
eq rand(fkvm4(P,S,B,HC)) = rand(P) .
-- for action fkvm5
ceq nw(fkvm5(P,S,B,HN)) = vm(is,S,B,cl(h(com(S,B,HN))),sig(sk(is),h(com(S,B,HN)))) , nw(P)
    if HN \in hbans(nw(P)) .
ceq nw(fkvm5(P,S,B,HN)) = nw(P)
    if not(HN \in hbans(nw(P))) .
eq rand(fkvm5(P,S,B,HN)) = rand(P) .
-- for action fkvm6
ceq nw(fkvm6(P,S,N,R))
    = vm(is,S,b(N),cl(h(com(S,b(N),h(R,N)))),sig(sk(is),h(com(S,b(N),h(R,N))))) , nw(P)
    if N \in bans(nw(P)) and R \in rands(nw(P)) .
ceq nw(fkvm6(P,S,N,R)) = nw(P)
    if not(N \in bans(nw(P)) and R \in rands(nw(P))) .
eq rand(fkvm6(P,S,N,R)) = rand(P) .
-- for action fkpm1
ceq nw(fkpm1(P,B,S,EP,GB)) = pm(ib,B,S,EP,GB) , nw(P) if EP \in eslps(nw(P)) and GB \in sigbs(nw(P)) .
ceq nw(fkpm1(P,B,S,EP,GB)) = nw(P) if not(EP \in eslps(nw(P)) and GB \in sigbs(nw(P))) .
eq rand(fkpm1(P,B,S,EP,GB)) = rand(P) .
-- for action fkpm2
ceq nw(fkpm2(P,S,N,R,HC,GB)) = pm(ib,b(N),S,enc(pk(la),slp(HC,N,R)),GB) , nw(P)
    if N \in bans(nw(P)) and R \in rands(nw(P)) and HC \in hcoms(nw(P)) and GB \in sigbs(nw(P)) .
ceq nw(fkpm2(P,S,N,R,HC,GB)) = nw(P)
    if not(N \in bans(nw(P)) and R \in rands(nw(P)) and HC \in hcoms(nw(P)) and GB \in sigbs(nw(P))) .
eq rand(fkpm2(P,S,N,R,HC,GB)) = rand(P) .
-- for action fkpm3
ceq nw(fkpm3(P,S,N,R,GB)) = pm(ib,b(N),S,enc(pk(la),slp(h(com(S,b(N),h(R,N))),N,R)),GB) , nw(P)
    if N \in bans(nw(P)) and R \in rands(nw(P)) and GB \in sigbs(nw(P)) .
ceq nw(fkpm3(P,S,N,R,GB)) = nw(P)
    if not(N \in bans(nw(P)) and R \in rands(nw(P)) and GB \in sigbs(nw(P))) .
eq rand(fkpm3(P,S,N,R,GB)) = rand(P) .
-- for action fkpm4
ceq nw(fkpm4(P,B,S,EP,HC))
    = pm(ib,B,S,EP,sig(sk(ib),EP,HC)) , nw(P) if EP \in eslps(nw(P)) and HC \in hcoms(nw(P)) .
ceq nw(fkpm4(P,B,S,EP,HC)) = nw(P) if not(EP \in eslps(nw(P)) and HC \in hcoms(nw(P))) .
eq rand(fkpm4(P,B,S,EP,HC)) = rand(P) .
```

**Figure 8. Equations to define transition rules corresponding to faking messages (1)**

```
-- for action fkpm5
ceq nw(fkpm5(P,B,S,EP,HN)) = pm(ib,B,S,EP,sig(sk(ib),EP,h(com(S,B,HN)))) , nw(P)
    if EP \in eslps(nw(P)) and HN \in hbans(nw(P)) .
ceq nw(fkpm5(P,B,S,EP,HN)) = nw(P)
   if not(EP \in eslps(nw(P)) and HN \in hbans(nw(P))) .
eq rand(fkpm5(P,B,S,EP,HN)) = rand(P) .
-- for action fkpm6
ceq nw(fkpm6(P,S,EP,N,R)) = pm(ib,b(N),S,EP,sig(sk(ib),EP,h(com(S,b(N),h(R,N))))) , nw(P)
    if EP \in eslps(nw(P)) and N \in bans(nw(P)) and R \in rands(nw(P)) .
ceq nw(fkpm6(P,S,EP,N,R)) = nw(P)
   if not(EP \in eslps(nw(P)) and N \in bans(nw(P)) and R \in rands(nw(P))) .
eq rand(fkpm6(P,S,EP,N,R)) = rand(P) .
-- for action fkpm7
ceq nw(fkpm7(P,S,N,R,HC))
    = pm(ib,b(N),S,enc(pk(la),slp(HC,N,R)),sig(sk(ib),enc(pk(la),slp(HC,N,R)),HC)) , nw(P)
    if N \in bans(nw(P)) and R \in rands(nw(P)) and HC \in hcoms(nw(P)) .
ceq nw(fkpm7(P,S,N,R,HC)) = nw(P)
    if not(N \in bans(nw(P)) and R \in rands(nw(P)) and HC \in hcoms(nw(P))) .
eq rand(fkpm7(P,S,N,R,HC)) = rand(P) .
-- for action fkpm8
ceq nw(fkpm8(P,S,N,R))
    = pm(ib,b(N),S,enc(pk(la),slp(h(com(S,b(N),h(R,N))),N,R)),
                  sig(sk(ib),enc(pk(la),slp(h(com(S,b(N),h(R,N))),N,R)),h(com(S,b(N),h(R,N))))) , nw(P)
    if N \in bans(nw(P)) and R \in rands(nw(P)) .
ceq nw(fkpm8(P,S,N,R)) = nw(P)
    if not(N \in bans(nw(P)) and R \in rands(nw(P))) .
eq rand(fkpm8(P,S,N,R)) = rand(P) .
-- for action fkqm1
ceq nw(fkqm1(P,S,HC,EP,GT,GB)) = qm(is,S,la,cl(HC),EP,GT,GB) , nw(P)
    if HC \in hcoms(nw(P)) and EP \in eslps(nw(P)) and GT \in sigs2s(nw(P)) and GB \in sigbs(nw(P)) .
ceq nw(fkqm1(P,S,HC,EP,GT,GB)) = nw(P) if not(HC \in hcoms(nw(P)) and EP \in eslps(nw(P)) and
                                               GT \in sigs2s(nw(P)) and GB \in sigbs(nw(P))) .
eq rand(fkqm1(P,S,HC,EP,GT,GB)) = rand(P) .
-- for action fkqm2
ceq nw(fkqm2(P,S,B,HN,EP,GT,GB)) = qm(is,S,la,cl(h(com(S,B,HN))),EP,GT,GB) , nw(P)
    if HN \in hbans(nw(P)) and EP \in eslps(nw(P)) and GT \in sigs2s(nw(P)) and GB \in sigbs(nw(P)) .
ceq nw(fkqm2(P,S,B,HN,EP,GT,GB)) = nw(P) if not(HN \in hbans(nw(P)) and EP \in eslps(nw(P)) and
                                                GT \in sigs2s(nw(P)) and GB \in sigbs(nw(P))) .
eq rand(fkqm2(P,S,B,HN,EP,GT,GB)) = rand(P) .
-- for action fkqm3
ceq nw(fkqm3(P,S,N,R,GT,GB))
    = qm(is,S,la,cl(h(com(S,b(N),h(R,N)))),enc(pk(la),slp(h(com(S,b(N),h(R,N))),N,R)),GT,GB) , nw(P)
    if N \in bans(nw(P)) and R \in rands(nw(P)) and GT \in sigs2s(nw(P)) and GB \in sigbs(nw(P)) .
ceq nw(fkqm3(P,S,N,R,GT,GB)) = nw(P)
    if not(N \in bans(nw(P)) and R \in rands(nw(P)) and GT \in sigs2s(nw(P)) and GB \in sigbs(nw(P))) .
eq rand(fkqm3(P,S,N,R,GT,GB)) = rand(P) .
-- for action fkqm4
ceq nw(fkqm4(P,S,HC,EP,GB)) = qm(is,S,la,cl(HC),EP,sig(sk(is),HC,EP),GB) , nw(P)
    if HC \in hcoms(nw(P)) and EP \in eslps(nw(P)) and GB \in sigbs(nw(P)) .
ceq nw(fkqm4(P,S,HC,EP,GB)) = nw(P)
    if not(HC \in hcoms(nw(P)) and EP \in eslps(nw(P)) and GB \in sigbs(nw(P))) .
  eq rand(fkqm4(P,S,HC,EP,GB)) = rand(P) .
-- for action fkqm5
ceq nw(fkqm5(P,S,B,HN,EP,GB))
    = qm(is,S,la,cl(h(com(S,B,HN))),EP,sig(sk(is),h(com(S,B,HN)),EP),GB) , nw(P)
    if HN \in hbans(nw(P)) and EP \in eslps(nw(P)) and GB \in sigbs(nw(P)) .
ceq nw(fkqm5(P,S,B,HN,EP,GB)) = nw(P)
    if not(HN \in hbans(nw(P)) and EP \in eslps(nw(P)) and GB \in sigbs(nw(P))) .
eq rand(fkqm5(P,S,B,HN,EP,GB)) = rand(P) .
-- for action fkqm6
ceq nw(fkqm6(P,S,N,R,GB))
    = qm(is,S,la,cl(h(com(S,b(N),h(R,N)))),enc(pk(la),slp(h(com(S,b(N),h(R,N))),N,R)),
         sig(sk(is),h(com(S,b(N),h(R,N))),enc(pk(la),slp(h(com(S,b(N),h(R,N))),N,R))),GB) , nw(P)
    if N \in bans(nw(P)) and R \in rands(nw(P)) and GB \in sigbs(nw(P)) .
ceq nw(fkqm6(P,S,N,R,GB)) = nw(P)
    if not(N \in bans(nw(P)) and R \in rands(nw(P)) and GB \in sigbs(nw(P))) .
eq rand(fkqm6(P,S,N,R,GB)) = rand(P) .
```

**Figure 9. Equations to define transition rules corresponding to faking messages (2)**

```
-- for action fkqm7
ceq nw(fkqm7(P,S,HC,EP,GT)) = qm(is,S,la,cl(HC),EP,GT,sig(sk(ib),EP,HC)) , nw(P)
    if HC \in hcoms(nw(P)) and EP \in eslps(nw(P)) and GT \in sigs2s(nw(P)) .
ceq nw(fkqm7(P,S,HC,EP,GT)) = nw(P)
    if not(HC \in hcoms(nw(P)) and EP \in eslps(nw(P)) and GT \in sigs2s(nw(P))) .
eq rand(fkqm7(P,S,HC,EP,GT)) = rand(P) .
-- for action fkqm8
ceq nw(fkqm8(P,S,B,HN,EP,GT))
    = qm(is,S,la,cl(h(com(S,B,HN))),EP,GT,sig(sk(ib),EP,h(com(S,B,HN)))) , nw(P)
    if HN \in hbans(nw(P)) and EP \in eslps(nw(P)) and GT \in sigs2s(nw(P)) .
ceq nw(fkqm8(P,S,B,HN,EP,GT)) = nw(P)
    if not(HN \in hbans(nw(P)) and EP \in eslps(nw(P)) and GT \in sigs2s(nw(P))) .
eq rand(fkqm8(P,S,B,HN,EP,GT)) = rand(P) .
-- for action fkqm9
ceq nw(fkqm9(P,S,N,R,GT))
    = qm(is,S,la,cl(h(com(S,b(N),h(R,N)))),enc(pk(la),slp(h(com(S,b(N),h(R,N))),N,R)),
                GT,sig(sk(ib),enc(pk(la),slp(h(com(S,b(N),h(R,N))),N,R)),h(com(S,b(N),h(R,N))))) , nw(P)
    if N \in bans(nw(P)) and R \in rands(nw(P)) and GT \in sigs2s(nw(P)) .
ceq nw(fkqm9(P,S,N,R,GT)) = nw(P)
    if not(N \in bans(nw(P)) and R \in rands(nw(P)) and GT \in sigs2s(nw(P))) .
eq rand(fkqm9(P,S,N,R,GT)) = rand(P) .
-- for action fkqm10
ceq nw(fkqm10(P,S,HC,EP)) = qm(is,S,la,cl(HC),EP,sig(sk(is),HC,EP),sig(sk(ib),EP,HC)) , nw(P)
    if HC \in hcoms(nw(P)) and EP \in eslps(nw(P)) .
ceq nw(fkqm10(P,S,HC,EP)) = nw(P)
    if not(HC \in hcoms(nw(P)) and EP \in eslps(nw(P))) .
eq rand(fkqm10(P,S,HC,EP)) = rand(P) .
-- for action fkqm11
ceq nw(fkqm11(P,S,B,HN,EP)) = qm(is,S,la,cl(h(com(S,B,HN))),EP,sig(sk(is),h(com(S,B,HN)),EP),
                                                    sig(sk(ib),EP,h(com(S,B,HN)))) , nw(P)
    if HN \in hbans(nw(P)) and EP \in eslps(nw(P)) .
ceq nw(fkqm11(P,S,B,HN,EP)) = nw(P)
    if not(HN \in hbans(nw(P)) and EP \in eslps(nw(P))) .
eq rand(fkqm11(P,S,B,HN,EP)) = rand(P) .
-- for action fkqm12
ceq nw(fkqm12(P,S,N,R))
    = qm(is,S,la,cl(h(com(S,b(N),h(R,N)))),enc(pk(la),slp(h(com(S,b(N),h(R,N))),N,R)),
                sig(sk(is),h(com(S,b(N),h(R,N))),enc(pk(la),slp(h(com(S,b(N),h(R,N))),N,R))),
                sig(sk(ib),enc(pk(la),slp(h(com(S,b(N),h(R,N))),N,R)),h(com(S,b(N),h(R,N))))) , nw(P)
    if N \in bans(nw(P)) and R \in rands(nw(P)) .
ceq nw(fkqm12(P,S,N,R)) = nw(P)
    if not(N \in bans(nw(P)) and R \in rands(nw(P))) .
eq rand(fkqm12(P,S,N,R)) = rand(P) .
-- for action fksm1
ceq nw(fksm1(P,S,C,GA)) = sm(ia,la,S,C,GA) , nw(P) if C \in rcodes(nw(P)) and GA \in sigas(nw(P)) .
ceq nw(fksm1(P,S,C,GA)) = nw(P) if not(C \in rcodes(nw(P)) and GA \in sigas(nw(P))) .
eq rand(fksm1(P,S,C,GA)) = rand(P) .
-- for action fksm2
ceq nw(fksm2(P,S,C,HC)) = sm(ia,la,S,C,sig(sk(ia),C,HC)) , nw(P)
    if C \in rcodes(nw(P)) and HC \in hcoms(nw(P)) .
ceq nw(fksm2(P,S,C,HC)) = nw(P) if not(C \in rcodes(nw(P)) and HC \in hcoms(nw(P))) .
eq rand(fksm2(P,S,C,HC)) = rand(P) .
-- for action fksm3
ceq nw(fksm3(P,S,B,C,HN)) = sm(ia,la,S,C,sig(sk(ia),C,h(com(S,B,HN)))) , nw(P)
    if C \in rcodes(nw(P)) and HN \in hbans(nw(P)) .
ceq nw(fksm3(P,S,B,C,HN)) = nw(P)
    if not(C \in rcodes(nw(P)) and HN \in hbans(nw(P))) .
eq rand(fksm3(P,S,B,C,HN)) = rand(P) .
-- for action fksm4
ceq nw(fksm4(P,S,C,N,R)) = sm(ia,la,S,C,sig(sk(ia),C,h(com(S,b(N),h(R,N))))) , nw(P)
    if C \in rcodes(nw(P)) and N \in bans(nw(P)) and R \in rands(nw(P)) .
ceq nw(fksm4(P,S,C,N,R)) = nw(P)
    if not(C \in rcodes(nw(P)) and N \in bans(nw(P)) and R \in rands(nw(P))) .
eq rand(fksm4(P,S,C,N,R)) = rand(P) .
```

**Figure 10. Equations to define transition rules corresponding to faking messages (3)**