FORMAL ANALYSIS OF A FAIR PAYMENT PROTOCOL

Jan Cederquist and Muhammad Torabi Dashti
CWI, Postbus 94079, 1090 GB Amsterdam, The Netherlands
{Cederqui,Dashti}@cwi.nl

Abstract We formally specify a payment protocol described in [Vogt et al., 2001]. This protocol is intended for fair exchange of time-sensitive data. Here the μCRL language is used to formalize the protocol. Fair exchange properties are expressed in the regular alternation-free μ-calculus. These properties are then verified using the finite state model checker from the CADP toolset. Proving fairness without resilient communication channels is impossible. We use the Dolev-Yao intruder, but since the conventional Dolev-Yao intruder violates this assumption, it is forced to comply to the resilient communication channel assumption.

1. Introduction

A fair exchange protocol aims at exchanging items in a *fair* manner. Informally, fair means that all involved parties receive a desired item in exchange for their own, or neither of them does so. It has been shown that fair exchange is impossible without a trusted third party [Pagnia and Gärtner, 1999]. A protocol for fair exchange of money for an item using customer's smart card as a trusted party is described in [Vogt et al., 2001]. This protocol considers time-sensitive items and is adapted for wireless and mobile applications which lack a reliable communication channel. Here a version of that protocol is considered. We describe, in contrast to [Vogt et al., 2001], the exact contents of all messages. The protocol is formally specified and the fairness properties are verified using a finite-state model checker.

In comparison to other security issues, such as secrecy and authenticity, fairness has not been studied formally so intensively. There are however some notable exceptions. [Shmatikov and Mitchell, 2002] use the finite state model checker Murφ to analyze fair exchange and contract signing protocols. They use an external intruder, based on the Dolev-Yao intruder, that collaborates with one of the participants to model the

malicious participant. Liveness can in general not be expressed in the Murφ language. Most fair-exchange properties can however be expressed as safety properties. But this is not the case with termination (of protocol). Termination thus relies on other arguments than a verification using Murφ. [Kremer and Raskin, 2001] use a game based approach for verifying non-repudiation and fair exchange. They use *alternating transition systems* (ATS) to model protocols and *alternating temporal logic* (ATL) to express the requirements. The method is automated using the model checker Mocha. They have no explicit intruder. Instead different versions of players are considered; honest and arbitrary. ATL then offers very neat ways of expressing all desired requirements, including liveness under fairness constraints. In ATS all players follow predetermined finite sequences of steps, including intruders (arbitrary versions of players). However, describing an intruder in such a way is not practical for large protocols. In [Schneider, 1998] a non-repudiation protocol is modeled using CSP, and proofs are generated by hand. Belief logic is used to formalize a protocol in [Zhou and Gollmann, 1998] and it is discussed what may be needed for the verification of non-repudiation protocols. In [Bella and Paulson, 2001] the theorem prover Isabelle is used to model a non-repudiation protocol by an inductive definition and to prove some desired properties.

In our work we formally specify a payment protocol in the process algebraic language μCRL [Groote and Ponse, 1995]. The idea of this protocol comes from [Vogt et al., 2001], but there are some differences (see section 6). Fairness properties for this protocol are formulated in the regular alternation-free μ-calculus [Mateescu, 2000] and verified using the model checker EVALUATOR 3.0 [Mateescu, 2000] from the CADP tool set [Fernandez et al., 1996].

Our formalization in μCRL contains a Dolev-Yao intruder [Dolev and Yao, 1983]. The intruder is not separated from malicious participants. Instead, we consider different versions of participants, honest and malicious, where a malicious participant is an intruder that has access to the participant's private key. Some fairness properties are liveness properties and to prove liveness properties resilient communication channels are needed (i.e. sent messages will eventually be delivered). Since the Dolev-Yao intruder has complete control over network, some cooperation from the intruder is needed when verifying liveness properties. This cooperation is obtained using fairness constraints on the labelled transition system generated from the protocol specification in μCRL.

The rest of the paper is organized as follows. In section 2 we give an overview of properties for fair exchange protocols. The fair exchange protocol we investigate is described in section 3. In section 4 the formal

analysis is described. Here the intruder model is presented and all properties verified using the model checker are given. Included in section 4 is also a brief description of some optimization techniques used to generate the state spaces. In section 5 the protocol is used in a practical context. Some concluding remarks are given in section 6.

Due to space constraints, most of our formalization of the protocol in μCRL cannot be presented in this paper. However, the complete formalization is given in [Cederquist and Dashti, 2004].

2. Fair Exchange Protocols

We assume two parties A and B. When the protocol starts, both parties have an item and they want to exchange items.

According to [Asokan, 1998], a fair exchange protocol is a protocol satisfying *effectiveness*, *fairness*, *timeliness* and *non-repudiability*. Effectiveness means that if both parties behave according to the protocol and none of them want to abort during the protocol round, then the protocol will terminate in a state where A has B's item and vice versa. An exchange protocol is called fair if, when it has terminated, either A has received B's item and B has received A's item, or none of the parties have lost their items. Timeliness means that the protocol will terminate for all parties (that behave according to the protocol) and after the termination point the degree of achieved fairness will not change. Non-repudiability is, in general, not considered as a primary requirement for fair exchange protocols, and it is omitted here.

[Asokan, 1998] distinguishes between *strong* and *weak* fairness. Strong fairness is the fairness described above. Weak fairness means that either strong fairness is achieved, or it is possible for a participant to prove to an outside party that an unfair situation has occurred. [Pagnia et al., 2003] extend Asokan's definitions by considering the parties' willingness to cooperate and compensation for suffered disadvantage.

3. Protocol Description

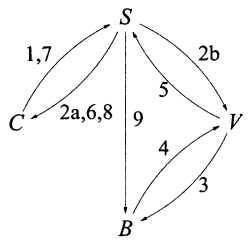
Here we describe the protocol which is to be analyzed in section 4. The protocol aims at fair exchange of time-sensitive data for some amount of money, between a customer (C) and a vendor (V). The exchange uses a bank (B) as a trusted online payment system and a trusted smartcard (S) attached to C. S is a tamper-proof hardware. The identity of S is however not necessarily known by V. Moreover, C is assumed to have a secure communication channel with S. When the protocol starts, V has an item m and a description $h(m)$ of m is known publicly. C wants to buy m for the amount a. Note that the item m is assumed to be

confidential and should not be revealed to untrusted parties unless they pay for it. Below we describe the intended scenarios of the protocol, when all participants are honest.

In the protocol description, $pay(C, V, a)$ means that C shall pay the amount a to V, $(m)_X$ is the notation for the message m signed by X (using X's private key), and $\{m\}_X$ is the notation for m encrypted for X (using X's public key). It is assumed that m comes along with $(m)_X$ and can be extracted by anyone. For an encrypted message $\{m\}_X$ only X can extract m. A publicly known hash function h is used for describing items and payments. T and F are two symbols. By convention we use T for positive responses and F for negative.

The main scenario (when none of the participants want to abort the protocol) is described as follows:

1.	$C \to S$:	$pay(C, V, a), h(m)$
	S :	$initiate(n)$
[2a.	$S \to C$:	$(h(m), t, n, v, a)_S$]
2b.	$S \to V$:	$h(m), (pay(C, V, a), n)_S$
3.	$V \to B$:	$(pay(C, V, a), n)_S$
	B :	$block(n)$
4.	$B \to V$:	$(\mathsf{T}, h((pay(C, V, a), n)_S))_B$
	V :	$commit(n)$
5.	$V \to S$:	$\{(m, n)_V\}_S$
6.	$S \to C$:	T, n
7.	$C \to S$:	T, n
	S :	$receive(n)$
8.	$S \to C$:	m
9.	$S \to B$:	$(n, \mathsf{T})_S$
	B :	$transfer(n)$, $terminate(n)$.



(1) C sends a query to S for buying item m from V for amount a. On this request, S generates a fresh nonce n. In this way, a protocol session possesses a unique nonce. Implicitly, S also notes the time t. (2a) S sends the nonce associated to the request and time to C. Later on, in step 6, when S asks C if the item is still interesting, it just needs to send the nonce. This simplifies the formalization. The time information is signed by S to prevent C from changing it. (We abstract away from this step in the formalization.) (2b) S signs and forwards the request together with the nonce to V. Since S is trusted, this message will be sent only upon a request from C. (3) If V wants to sell m to C for price a, it forwards the request to B. B notices the signature of S, checks whether the nonce n is fresh and that C has the amount a in its account. If this is the case, the money is blocked on C's account. (4) B notifies V that a transfer of amount a from C's account to V's account is possible. After this step V knows S is trusted. (5) V informs S that C

can buy m for the amount a (n refers to a). (6) S validates the received item by comparing it with $h(m)$ and asks if C is still interested in the item. (7) If C still wants the item, it answers T. (8) S sends the item m to C. (9) S asks B to transfer the money, that was blocked on C's account, to V's account. On this request B performs the transaction.

(The "abstract" actions *initiate*(n), *block*(n), *receive*(n), *transfer*(n) and *terminate*(n) are explained with more details in section 4.4, and so are the actions *unblock*(n) and *cancel*(n) below.)

There are some alternative scenarios of the protocol. When B receives a payment request, the nonce n may not be fresh or C may not have the required amount of money on its account:

$$\begin{array}{lll} 4^1. & B \to V: & (\mathsf{F}, h((pay(C,V,a),n)_S))_B \\ 5^1. & V \to S: & (n)_V \\ 6^1. & S \to C: & \mathsf{F}, n \\ 7^1. & S \to B: & (n,\mathsf{F})_S \\ & B: & unblock(n),\ terminate(n). \end{array}$$

(7^1) S asks B to unblock money at C's account. If the money was blocked earlier, with the same nonce, B unblocks it. If V does not want to sell m to C for the amount a, step 5^1 follows immediately after step $2b$.

After step 2 and before step 6 (6^1), C has the possibility to cancel the payment. This prevents V from blocking C's money without sending the item to S:

$$\begin{array}{lll} & C: & cancel(n) \\ 6^2. & C \to S: & n \\ 7^2. & S \to B: & (n,\mathsf{F})_S \\ 8^2. & S \to V: & (\mathsf{F},n)_S \\ & B: & unblock(n),\ terminate(n). \end{array}$$

S erases the session information after sending *unblock* (or *transfer*) commands to B, and does not consider any message with a nonce from completed sessions.

In exchange of items whose value may change during time, the protocol provides a possibility for C to reject items in case of (intentional) delay in delivery. So, C can answer F after step 6:

$$\begin{array}{lll} 7^3. & C \to S: & \mathsf{F}, n \\ 8^3. & S \to B: & (n,\mathsf{F})_S \\ & B: & unblock(n),\ terminate(n). \end{array}$$

After step $2b$, S can perform a *timeout*:

$$\begin{array}{lll} & S: & timeout \\ 3^4. & S \to C: & \mathsf{F}, n \\ 4^4. & S \to V: & (\mathsf{F},n)_S \\ 5^4. & S \to B: & (n,\mathsf{F})_S \\ & B: & unblock(n),\ terminate(n). \end{array}$$

The timeout forces a time limit on the steps 2*b*–7, it prevents in particular C from waiting arbitrarily before answering in step 7. Concerning timeout, our description is non-deterministic. But it can also be assumed that S reads the start time t, that was sent to C in step 2*a*, and that it has a limit Δt hard coded or provided by V. If the current time is greater than $t + \Delta t$, it generates a timeout.

4. Formal Analysis

The formalization of the protocol described in section 3 is carried out in μCRL [Groote and Ponse, 1995]. The μCRL toolset includes an automatic state space (labelled transition systems) generator and symbolic state space reduction tools. The properties effectiveness, timeliness and fairness are expressed in the regular alternation-free μ-calculus [Mateescu, 2000]. The model checker EVALUATOR 3.0 [Mateescu, 2000] from the CADP tool set [Fernandez et al., 1996] is then used to verify these properties (the formulas 1 to 11, in the sections 4.4 to 4.7).

For fair exchange protocols, beside protection from external intruders, the participants need to be protected from each other. In our formal model(s), we have three cases: (i) both C and V behave according to the protocol, (ii) C is malicious (C is the attacker) and (iii) V is malicious (V is the attacker). In the cases (ii) and (iii) all messages go via the attacker, with exception of the messages between C and S, which are sent over a secure link. When verifying effectiveness, case (i) is considered. All other properties are verified for the cases (ii) and (iii).

4.1 The μCRL specification language

Here we briefly describe the symbols used in the formalization below. For a complete description of the syntax and semantics of μCRL we refer to [Groote and Ponse, 1995].

The symbols . and + are used for the sequential and alternative composition operator, respectively. The operator $\sum_{d \in D} P(d)$ behaves like $P(d_1) + P(d_2) + \cdots$. The process expression **if** b **then** p **else** q, where b is a term of sort **bool** and, p and q are processes, behaves like p if b is true, and like q if b is false. Finally, the constant δ expresses that, from now on, no action can be performed.

The notations $send(a, x, b)$ and $recv(a, x, b)$ are used for the actions "A sends message x to B" and "B receives message x from A", respectively. In our model, *send* and *recv* actions are synchronized, i.e. A can only perform $send(a, x, b)$ if B at the same time performs $recv(a, x, b)$ and vice versa. This synchronization point is denoted $com(a, x, b)$ (in section 3, the notation $A \rightarrow B : x$ was used for that).

4.2 Regular Alternation-free μ-calculus

The regular alternation-free μ-calculus is used here to formulate properties of (states in) labelled transition systems (see the sections 4.4–4.7). It is a fragment of μ-calculus that can be efficiently checked. Here we just briefly describe what is needed for expressing the fairness properties of the protocol we investigate. For a complete description of the syntax and semantics we refer to [Mateescu, 2000]. The regular alternation-free μ-calculus is built up from three types of formulas: *action formulas*, *regular formulas* and *state formulas*. We use '.', '$\vee$' and '*' for concatenation, choice and transitive-reflexive closure, respectively, for regular formulas. $\mathcal{F}$ and $\mathcal{T}$ are used in both action formulas and state formulas. In action formulas they represent *no action* and *any action*, respectively. The meaning of $\mathcal{F}$ and $\mathcal{T}$ in state formulas are the empty set and the entire state space, respectively. The operators $\langle \cdots \rangle$ and $[\cdots]$ have their usual meaning ($\Diamond$ and $\Box$ in modal logics). Finally, μ is the minimal fixed point operator.

4.3 Intruder Models

We consider the Dolev-Yao intruder [Dolev and Yao, 1983]. It can remember all messages that have been transmitted over network. It can decrypt and sign messages, if it knows the corresponding key. It can compose new messages from its knowledge. It can also remove or delay messages in favour of others being communicated.

Below we define two intruder models in μCRL, I and I'. Both of them are equivalent to the Dolev-Yao intruder, but they behave differently under fairness constraints[1]. I is used when verifying safety properties and I' when verifying liveness properties. The reason for using both of them is that I is not suitable for liveness properties, and I' is expensive to use when generating state spaces (see section 4.8).

The intruder I acts as customer (or vendor), intruder and network. All messages (x) are sent to I explicitly. I decomposes (*decomp*) the messages and adds the pieces to its knowledge (X). I then uses its knowledge to synthesize (*synth*) new messages. How well the decomposition and the synthesis work depend on what private keys the intruder knows (abilities to sign and decrypt messages), *decomp* and *synth* are thus parameterized over users whose private keys are known. (For effi-

[1]We are using two notions of fairness; fairness of a protocol and fairness constraints of a labelled transition system. The second one is used to describe "fair" execution traces. In our case, a trace is fair when no possibilities are excluded forever. Then only fair execution traces are considered when proving the desired (liveness) property. To avoid confusion, we refer to these two notions as "fairness" and "fairness constraints".

ciency reasons the union $\bigcup$ also depends on known private keys.)

$$
\begin{aligned}
I(X) = (\ & \textstyle\sum_{p \in Agent, x \in Message} \\
& \quad recv(p, x, i).I(X \textstyle\bigcup_i decomp_i(x)) + \\
& \quad \textbf{if } synth_i(x, X) \\
& \quad \textbf{then } send(i, x, p).I(X) \\
& \quad \textbf{else } \delta\ ) + \\
(\ & \textstyle\sum_{m \in Item} \\
& \quad \textbf{if } synth_i(m, X) \\
& \quad \textbf{then } got\text{–}hold\text{–}of(m).I(X) \\
& \quad \textbf{else } \delta\ )
\end{aligned}
$$

In order to prove liveness properties, resilient communication channels are assumed. In fact, without this assumption fair exchange is not possible, because then the attacker can simply choose to never send the item to one of the participants. In the presence of an intruder, resilient communication channels are obtained by imposing fairness constraints on the labelled transition system generated from the protocol specification. These fairness constraints are expressed directly in regular μ-calculus formulas (see property 7 in section 4.6). The use of fairness constraints makes the model checker "skip circuits" and, in particular, it eventually forces the intruder to try to synthesize and send messages whenever there is a recipient. Some amount of cooperation from the intruder is usually needed in order to prove liveness properties. But, the fact that the intruder I does not forget anything and its abilities to construct messages itself together with fairness constraints can make "too many" liveness properties true. In fact, an erroneous protocol that does not terminate without intruder, may terminate with the intruder I and fairness constraints.

The second intruder I' can also synthesize new messages from its knowledge, but it is not forced to. However, using fairness constraints, it is forced to comply to the resilient communication channel assumption. It is parameterized over a set of "resilient links" and all messages sent over these links should eventually be delivered. In our case the resilient link is the link between S and B. The corresponding messages are represented by the set Z. As I, I' gathers a set X of knowledge by intercepting all communications. But, it can explicitly forget pieces from this knowledge. The intruder uses a separate buffer Y of messages transmitted over the resilient links. When fairness constraints are used, the intruder is forced to eventually send all messages from this buffer. The resilient channel assumption will thus be preserved. Since I' can forget, it does not have to generate new messages. However, this does not restrict the intruder's power in general, as it has the choice of keeping

its knowledge as well.

$$
\begin{aligned}
I'(X,Y) = (\ & \textstyle\sum_{p \in Agent, x \in Message} \\
& \quad recv(p,x,i). \\
& \quad \textbf{if } x \in Z \\
& \quad \textbf{then } I'(decomp_i(x) \textstyle\bigcup_i X, insert(x,Y)) \\
& \quad \textbf{else } I'(decomp_i(x) \textstyle\bigcup_i X, Y)\ ) + \\
(\ & \textstyle\sum_{x \in Message} \\
& \quad \textbf{if } x \in X \\
& \quad \textbf{then } I'(X \setminus \{x\}, Y) \\
& \quad \textbf{else } \delta\ ) + \\
(\ & \textstyle\sum_{x \in Message, p \in Agent} \\
& \quad \textbf{if } x \in Y \vee synth_i(x,X) \\
& \quad \textbf{then } send(i,x,p).I'(X, remove(x,Y)) \\
& \quad \textbf{else } \delta\ )
\end{aligned}
$$

4.4 Abstract Actions

The CADP toolset [Fernandez et al., 1996] that we use to analyze the labelled transition system generated from a μCRL specification does not allow variables in action parameters, in regular μ-calculus formulas. So, properties containing variables should actually be checked for each instance. To avoid this, the protocol is extended with abstract actions (*initiate*(n), *block*(n), *receive*(n), *transfer*(n),...) that can be used instead of actions containing more variables. In fact, each protocol session is associated to a nonce, so abstract actions (which only contain nonces) are enough for expressing most interesting properties of the protocol. Besides, they highlight implicit steps in the protocol and render more readable properties.

For termination, the "abstract" action *terminate*(n) in B (where n is a nonce) is used, instead of actual termination points for the users (C and V). This action is used because it is convenient to abstract away from messages to the users saying that a protocol round is terminated. This abstraction is safe since, if such messages had been used, the resilient communication channel assumption would have guaranteed their delivery. Thus *terminate*(n) implies that the users terminate. Also note, the protocol may continue after *terminate*(n) with another protocol round, using another (fresh) nonce.

The meaning of some of the other abstract actions need to be uniquely defined. We start with *block*(n). Without loss of generality we can assume that *block*(n) happens at the same time as (or immediately after) B receives $(pay(C,V,a),n)_S$. So, *block*(n) can be defined as *the amount a is blocked* (for nonce n). The fact that a indeed is the correct amount

follows from

$$[\mathcal{T}^*.com(s,(h(m),(pay(C,V,a_1),n)_S),i).\\ \mathcal{T}^*.com(i,(pay(C,V,a_2),n)_S,b)]\mathcal{F}, \tag{1}$$

where a_1 and a_2 are different amounts, and i is either c or v, depending on who is malicious. Property 1 thus says that the intruder cannot change the amount that C is willing to pay. We define $transfer(n)$ and $unblock(n)$ to mean that the amount, which was blocked in $block(n)$, is transfered and unblocked, respectively. Now we turn to $receive(n)$. It can be assumed that $receive(n)$ happens at the same time as S sends an item m to C. The fact that this item is the correct item (the item C ordered) follows from

$$[\mathcal{T}^*.com(c,(pay(C,V,a),h(m_1)),s).initiate(n).\\ \mathcal{T}^*.receive(n).com(s,m_2,c)]\mathcal{F}, \tag{2}$$

where m_1 and m_2 are different items.

A malicious customer could possibly get hold of an item m by other means than from S in action $com(s,m,c)$. To show that this is not the case, we verify

$$[(\neg com(s,m,c))^*.got\text{–}hold\text{–}of(m)]\mathcal{F}, \tag{3}$$

where $got\text{–}hold\text{–}of(m)$ is an abstract action that occur if the malicious customer manages to synthesize the item m from gained knowledge (see definition of intruder I, section 4.3).

4.5 Effectiveness

For effectiveness all participants are assumed to be honest and none of them want to abort the protocol. First, termination is inevitable

$$[\mathcal{T}^*.initiate(n)]\mu X(\langle\mathcal{T}\rangle\mathcal{T}\wedge[\neg terminate(n)]X), \tag{4}$$

for an arbitrary nonce n. Second, if S does not timeout, V does not say that C cannot buy the item, C does not answer $\mathcal{F}$ when S asks if the item is still valuable, and C does not cancel the payment, then the money will be transfered to V upon termination:

$$[(\neg(timeout\vee com(v,(n)_v,s)\vee com(c,(\mathcal{F},n),s)\vee cancel(n)\vee\\ transfer(n)))^*.terminate(n)]\mathcal{F}. \tag{5}$$

Under the same conditions, the item will also be received:

$$[(\neg(timeout\vee com(v,(n)_v,s)\vee com(c,(\mathcal{F},n),s)\vee cancel(n)\vee\\ receive(n)))^*.terminate(n)]\mathcal{F}. \tag{6}$$

4.6 Timeliness

First, we verify that each fair trace eventually reaches $terminate(n)$. Whenever $terminate(n)$ has not occurred, there is a path leading to $terminate(n)$:

$$[\mathcal{T}^*.initiate(n).(\neg terminate(n))^*]\langle\mathcal{T}^*.terminate(n)\rangle\mathcal{T}. \quad (7)$$

Second, the degree of fairness does not change after termination:

$$\begin{array}{l}[\mathcal{T}^*.terminate(n).\mathcal{T}^*. \\ (receive(n) \vee block(n) \vee unblock(n) \vee transfer(n))]\mathcal{F}.\end{array} \quad (8)$$

4.7 Fairness[2]

For the properties that guarantee fairness it is important that the protocol terminates, which is part of timeliness, section 4.6.

Here we split up the notion of fairness (introduced in section 2) into fairness for C and V individually. We say that the protocol is fair for C if, whenever C pays for an item, C will receive it (V potentially being malicious). Fairness for V is defined correspondingly. Fairness for C is thus formalized as

$$[(\neg receive(n))^*.transfer(n).(\neg receive(n))^*.terminate(n)]\mathcal{F}. \quad (9)$$

From C's point of view it is also important that if money for an item is blocked and C does not receive the item, the block will be removed. The following property may thus also be considered as fairness for C:

$$[\mathcal{T}^*.block(n).(\neg(receive(n) \vee unblock(n)))^*.terminate(n)]\mathcal{F}. \quad (10)$$

Fairness for V means that if an item (corresponding to the nonce n) is received, money will be transfered:

$$[(\neg transfer(n))^*.receive(n).(\neg transfer(n))^*.terminate(n)]\mathcal{F}. \quad (11)$$

4.8 Model Checking Details

One of the major obstacles during this work was state space explosion. The case when the customer is malicious turned out to be most difficult to generate. Our experiments show that this is mainly due to different knowledges of the intruder, gathered during different execution traces. A

[2]The notion of fairness we are proving corresponds to $\boldsymbol{F_5}$ in the hierarchy of fairness guarantees described in [Pagnia et al., 2003].

common abstraction technique in such situations is to make the knowledge of the intruder more uniform. We do that explicitly by, at the end of a protocol round, giving the intruder information about traces that were not taken. More traces will now end up in same states, with a smaller state space as result. This extra information for the intruder should be chosen carefully though to avoid "false attacks". For safety properties this technique is sound, since the intruder just becomes more powerful. This technique may however make "too many" liveness properties true. When assuming fairness constraints, an intruder with more knowledge provides more possibilities to reach a state. Instead, when generating the state space for proving liveness properties, we explicitly put a δ immediately after the action we want always to be reached. In this way, large parts of the state space will never be generated. In addition to these two methods, all actions except for the ones used in the properties are "hidden" and symbolic reduction techniques from the μCRL toolset (see [Blom et al., 2001] for a description of these techniques) are applied to reduce the state spaces.

Using the techniques described above we could generate the state spaces and prove the safety properties, with 3 nonces and 2 different items (also 1 nonce, 2 different items with 2 possible different prices), in the malicious customer and malicious vendor cases. For liveness properties (termination in case of malicious customer and vendor), it was impossible to consider concurrent sessions. The state spaces were generated for 2 items and 2 prices.

5. Practical Considerations

It is usually not possible for a smart card to receive large chunks of data, and store or process them. This limitation can cause practical problems when the item is some large software, for instance, and the smart card should store and validate it. On the other hand, if the item validation phase is removed, it is not clear how the vendor is prevented from sending fake items. Here we suggest employing a trusted offline Item Validation Party (IVP) that guarantees correspondence between an encrypted item and the description of the item. The protocol in section 3 is modified by adding two messages at the beginning:

$$\begin{array}{lll} 0a. & C \rightarrow V: & h(m) \\ 0b. & V \rightarrow C: & \{m\}_{pk}, (h(m), h(\{m\}_{pk}), pk)_{IVP}. \end{array}$$

In this scenario it is assumed that V generates key pairs (pk,sk) for encryption and decryption of items. There is also an IVP which validate encryptions offline. When C asks for an item with description $h(m)$, it will receive m encrypted with pk along with a certificate from IVP. In

this way, C can validate the item before decrypting it. Then C buys the decryption key sk from V using the protocol (in section 3), where the public key pk replaces the description of the item $h(m)$ (in message 1) and S checks that sk and pk match.

Assuming perfect cryptography and that the key pairs (pk,sk) are used only once, makes it safe to abstract away from the two initial messages $0a$ and $0b$. Consequently, correctness of this protocol follows from correctness of the protocol described in section 3, which was treated formally.

6. Conclusion

We have formally specified a payment protocol and verified its fairness properties. The idea of the protocol comes from [Vogt et al., 2001], but there are some differences. A version of the protocol that has an online payment system (bank) is considered. We implement revocable payments using *block*, *unblock* and *transfer* (as described in section 3). To protect the vendor from the customer being passive, the smartcard can *timeout* (in fact, without *timeout* property 7 does not hold). We have also relaxed the assumptions on the communication links (eavesdropping, replay and forging of messages is possible in our model).

We have implemented an intruder that, for safety properties, is equivalent to the Dolev-Yao intruder (which is the most powerful intruder, see [Cervesato, 2001]). Our intruder is particularly suitable for verifying liveness properties, since it does not violate the resilient communication channel assumption under fairness constraints. It can be used in general purpose specification languages like μCRL.

Acknowledgments

We are grateful to the anonymous referees for their constructive and detailed comments. We would also like to thank Stefan Blom, Wan Fokkink, Jaco van de Pol and Miguel Valero for discussions and comments on earlier versions of this paper.

The first author was supported by an ERCIM Fellowship.

References

[Asokan, 1998] Asokan, N. (1998). *Fairness in electronic commerce.* PhD thesis, University of Waterloo.

[Bella and Paulson, 2001] Bella, G. and Paulson, L. C. (2001). Mechanical proofs about a non-repudiation protocol. In Boulton, R. J. and Jackson, P. B., editors, *Theorem Proving in Higher Order Logics, 14th International Conference, TPHOLs 2001*, volume 2152 of *LNCS*, pages 91–104. Springer-Verlag.

[Blom et al., 2001] Blom, S., Fokkink, W., Groote, J. F., van Langevelde, I., Lisser, B., and van de Pol, J. (2001). μCRL: A toolset for analysing algebraic specifications. In *Proceedings of the 13th International Conference on Computer Aided Verification*, volume 2102 of *LNCS*, pages 250–254. Springer-Verlag.

[Cederquist and Dashti, 2004] Cederquist, J. and Dashti, M. (2004). Formal analysis of a fair payment protocol. Technical Report SEN-R0410, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

[Cervesato, 2001] Cervesato, I. (2001). The Dolev-Yao Intruder is the Most Powerful Attacker. In Halpern, J., editor, *16th Annual Symposium on Logic in Computer Science — LICS'01*, Boston, MA. IEEE Computer Society Press.

[Dolev and Yao, 1983] Dolev, D. and Yao, A. C. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208.

[Fernandez et al., 1996] Fernandez, J.-C., Garavel, H., Kerbrat, A., Mateescu, R., Mounier, L., and Sighireanu, M. (1996). CADP: A protocol validation and verification toolbox. In Alur, R. and Henzinger, T. A., editors, *Proceedings of the 8th Conference on Computer-Aided Verification*, volume 1102 of *LNCS*, pages 437–440. Springer-Verlag.

[Groote and Ponse, 1995] Groote, J. and Ponse, A. (1995). The syntax and semantics of μCRL. In Ponse, A., Verhoef, C., and van Vlijmen, S. F. M., editors, *Algebra of Communicating Processes '94*, Workshops in Computing Series, pages 26–62. Springer-Verlag.

[Kremer and Raskin, 2001] Kremer, S. and Raskin, J. (2001). A game-based verification of non-repudiation and fair exchange protocols. In Larsen, K. and Nielsen, M., editors, *Proceedings of the 12th International Conference on Concurrency Theory*, volume 2154 of *LNCS*, pages 551–565. Springer-Verlag.

[Mateescu, 2000] Mateescu, R. (2000). Efficient diagnostic generation for boolean equation systems. In *Proceedings of 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2000*, volume 1785 of *LNCS*, pages 251–265. Springer-Verlag.

[Pagnia and Gärtner, 1999] Pagnia, H. and Gärtner, F. C. (1999). On the impossibility of fair exchange without a trused third party. Technical Report TUD-BS-1999-02, Department of Computer Science, Darmstadt University of Technology.

[Pagnia et al., 2003] Pagnia, H., Vogt, H., and Gärtner, F. C. (2003). Fair exchange. *The Computer Journal*, 46(1):55–7.

[Schneider, 1998] Schneider, S. (1998). Formal analysis of a non-repudiation protocol. In *Proceedings of The 11th Computer Security Foundations Workshop*, pages 54–65. IEEE Computer Society Press.

[Shmatikov and Mitchell, 2002] Shmatikov, V. and Mitchell, J. C. (2002). Finite-state analysis of two contract signing protocols. *Theoretical Computer Sciene*, 283(2):419–450.

[Vogt et al., 2001] Vogt, H., Pagnia, H., and Gärtner, F. C. (2001). Using smart cards for fair exchange. In *Electronic Commerce – WELCOM 2001*, volume 2232 of *LNCS*, pages 101–113. Springer-Verlag.

[Zhou and Gollmann, 1998] Zhou, J. and Gollmann, D. (1998). Towards verification of non-repudiation protocols. In *International Refinement Workshop and Formal Methods Pacific '98: Proceedings of IRW/FMP '98, Discrete Mathematics and Theoretical Computer Science Series*, pages 370–380. Springer-Verlag.