Suitability of a classical analysis method for e-commerce protocols

Sigrid Gürgens[1] and Javier Lopez[2]

[1] Fraunhofer - Institute for Secure Telecooperation SIT
guergens@sit.fraunhofer.de
[2] Computer Science Department, University of Malaga
jlm@lcc.uma.es

Abstract. We present the adaptation of our model for the validation of key distribution and authentication protocols to address specific needs of protocols for electronic commerce. The two models defer in both the threat scenario and in the formalization. We demonstrate the suitability of our adaptation by analyzing a specific version of the Internet Billing Server protocol introduced by Carnegie Mellon University. Our analysis shows that, while the security properties a key distribution or authentication protocol shall provide are well understood, it is often not clear what properties an electronic commerce protocol can or shall provide. Our methods rely on automatic theorem proving tools. Specifically, we used "Otter", an automatic theorem proving software developed at Argonne National Laboratories.

Keywords: Security analysis, cryptographic protocol, automatic theorem proving, protocol validation, electronic commerce

1 Introduction

Cryptographic protocols play a key role in communication via open networks such as the Internet. These networks are insecure in the sense that any adversary with some technical background can monitor and alter messages interchanged by application users. Thus cryptographic protocols are used to provide the desired security properties. Messages are encrypted to ensure that only the intended recipient can understand them, digital signatures are applied to provide a proof that a message had been received, etc. All of these communication "services" can be provided as long as the cryptographic protocols themselves are correct and secure.

However, design of crytographic protocols is a difficult and error-prone task, and many of these largely used protocols have been shown to contain flaws. For this reason the use of formal methods that allow for the verification/validation of such protocols in a systematic and formal way has received increasing attention. In the last ten to fifteen years, active areas of research have developed around the problems of:

- Developing design methodologies which yield cryptographic protocols for which security properties can be formally proven.
- Formal specification and verification/validation of cryptographic protocols.

An early paper which addresses the first of these issues is [4]. Here, it is argued that protocols should not be defined simply as communicating programs but rather as sequences of messages with verifiable properties; i.e. security proofs can not be based on unverifiable assumptions about how an opponent constructs its messages. As with much of the research work done in the area of cryptography, this research does not adopt the framework of formal language specifications. Some other work along the lines of specifying provably secure protocols includes that of Bellare and Rogaway [3], Shoup and Rubin [28] (which extends the model of Bellare and Rogaway to smartcard protocols), Bellare, Canetti and Krawczyk [2], and the work of Heintze and Tygar considering compositions of cryptoprotocols [11].

Another approach worth mentioning is [26]. Here agents are modeled by using communicating automata and security properties are defined and can be formally proven in terms of properties of abstract communication channels.

The second issue, formal specification and automatic verification or validation methods for cryptographic protocols, has developed into a field of its own. Partial overviews of this area can be found in [16], [14] and [22].

Most of the work in this field can be categorized as either development of logics for reasoning about security (so-called authentication logics) or as development of model checking tools. Both techniques aim at verifying security properties of formally specified protocols.

As regarding the first category, a seminal paper on logics of authentication is [6]. Work in this area has produced significant results in finding protocol flaws, but also appears to have limitations which will be hard to overcome within the paradigm.

Model checking, on the other hand, involves the definition of a state space (typically modelling the "knowledge" of different participants in a protocol) and transition rules which define both the protocol being analyzed, the network properties, and the capabilities of an enemy. Initial work in this area can be traced to [9].

Much has already been accomplished. A well-known software tool is the NRL Protocol Analyzer [15], which has recently been expanded to include some specification capabilities, thus being able to specify the security requirements of the SET protocol being used by Mastercard and Visa [18]. Other notable work includes Lowe's use of the Failures Divergences Refinement Checker in CSP [13] and Schneider's use of CSP [27]. Recently, the CSP model was extended to cover non-repudiation, a security requirement of e-commerce protocols [25]. To do so, the outside intruder was removed from the model and the protocol participants were given the ability to fake messages. However, it is not clear whether this model can be extended to malicious agents with the ability to intercept messages as well.

Also to mention are the model checking algorithms of Marrero, Clarke, and Jha [14]; and Paulson's use of induction and the theorem prover Isabelle [22]. To the best of our knowledge, the model checking approach has been used almost exclusively for verification of cryptographic protocols.

Verification can be achieved efficiently if simplifying assumptions are made in order to obtain a sufficiently small state space. Verification tools which can handle infinite state spaces must simplify the notions of security and correctness to the point that proofs can be obtained using either induction or other logical means to reason about infinitely many states. Both methods have produced valuable insight into ways in which cryptographic protocols can fail.

The problem, however, is not only complicated but it is also evolving. The "classical" work has centered around proving security of key-distribution and authentication protocols, focusing on message properties such as key freshness, message confidentiality, and message origin authentication. Currently, with electronic commerce applications deployment, cryptographic protocols are being adapted for implementing commercial transactions.

This new generation of protocols imposes higher requirements on security issues. Secure electronic commerce functions such as transaction certification, electronic notary functions, operational performance, commerce disposal, anonymity, auditing, etc., are necessary for a successful deployment of electronic commerce transactions. These functions in turn produce requirements like identification of the sender and/or the receiver of a message, certification of delivery, confirmation of arrival, timestamping, detection of tampering, electronic storage of originality-guaranteed information, approval functions, and access control.

Consequently, protocols for electronic commerce scenarios are becoming much more complicated, and are more likely to contain errors. As a result, the complexity for their security analysis is increasing exponentially too. Moreover, electronic commerce transactions involve an increasing number of participants in one protocol run: Issuers, cardholders, merchants, acquirers, payment gateways and Certification Authorities (CAs). This again makes the security analysis problem harder to solve.

After Kailar's approach to analyze accountability in electronic commerce protocols [12] , there have been attempts to model other protocols such as Kerberos [1], TLS/SSL [23], Cybercash coin-exchange [5], and, as already mentioned, SET [18].

Already existing analysis techniques can be successfully adapted, and hence, applied to electronic commerce protocols (see [25]). In this paper we show how the method we have developed for the analysis of classical protocols can be adapted to cover specific needs of e-commerce protocols.

The rest of the paper is organized as follows: Section 2 describes the communication and attack model we developed for the analysis of protocols related to key distribution and authentication (which we will call "classical protocols" throughout this paper). Section 3 illustrates the way the theorem prover Otter is used for protocol analysis purposes, following criteria established in the previous section. In section 4 we present a new type of attack and discuss some known

attacks, in section 5 we explain how the model described in section 3 is adapted for the analysis of electronic commerce protocols. As an example, section 6 and 7 describe the IBS payment protocol and the results of our analysis, respectively. Section 8 concludes the paper.

2 The communication and attack model

The security analysis of protocols does not deal with weaknesses of the encryption and decryption functions used. In what follows we assume perfect encryption in the following sense:

1. Messages encrypted using a function f and a secret key K can only be decrypted with the inverse function and key f^{-1}, K^{-1}.
2. A key can not be guessed.
3. For the generation of a ciphertext $\{m\}_K$, it is necessary to know both m and K.

While the first two items describe generally accepted properties of encryption functions, the last one does not hold in general. Actually, to date no proof is known for a cryptoalgorithm to provide this property.

All communication is assumed to occur over insecure communication channels. We model these (in accordance with the Dolev-Yao attack model) by assuming that there is one more agent E that intercepts all messages sent by others. After intercepting, E can change the message to anything it can compute. This includes changing the destination of the message and the supposed identity of the sender. Furthermore, E can be considered a valid agent, thus may or may not also share a symmetric key K_{ES} with S and/or own a key pair (SK_E, PK_E) for an asymmetric algorithm, and can be allowed to initiate communication either with other agents or S. Our model leaves these decisions open to be fixed for a particular analysis run.

E intercepts all messages and the agents and S only receive messages sent by E. What E can send depends on what it is able to generate, and this in turn depends on what E knows. The knowledge of E can be recursively defined as follows:

1. Depending on the decision for a particular analysis run, E may or may not know the names of the agents.
2. E knows the keys it owns.
3. E knows its own random numbers.
4. E knows every message it has received.
5. E knows every part of a message received (where a ciphertext is considered one message part).
6. E knows everything it can generate by enciphering data it knows using data it knows as a key.
7. E knows every plaintext it can generate by deciphering a ciphertext it knows, provided it knows the inverse of both the key and the encryption function used as well.

8. E knows every concatenation of data it knows.
9. E knows the origin and destination of every message.
10. At every instance of the protocol run, E knows the "state" of every agent, i.e. E knows the next protocol step they will perform, the structure of a message necessary to be accepted, in principle what they will send after having received (and accepted) a message, etc.

At every state in a possible run, after having received an agent's message, E has the possibility to forward this message or to send whatever message it can generate to one of the agents or S. Obviously, the state space will grow enormously. Thus, we must limit the concept of knowledge if the goal is to produce a validation tool which can be used in practice. Therefore we restrict the definition in various ways in order to make E's knowledge set finite. For example, if the protocol to be analyzed uses a public key algorithm, we do not allow E to generate ciphertexts using data as a key which is neither the private nor the public part of a key pair. We do this on an ad-hoc basis. This problem needs further exploration. [1]

Another design desicion concerns the question of whether or not messages contain types, i.e. whether a recipient of a message can distinguish, say, a random number from a cryptographic key. There is an open discussion in the research community on whether protocol flaws resulting from type confusion are relevant. We believe that this question can not be decided a priori but is dependant on the message format of the particular application that the protocol is used in. It may be advantageous to know that a particular type check is not needed in order to achieve the desired security properties (by skipping unnecessary checks, performance can be improved), but it is absolutely necessary to know that a particular check is needed to avoid an attack in those applications that do not perform type checks by default. Consequently, our analysis model does provide the possibility to add types to messages, but for the analysis of classical protocols we assume that messages are not typed.

3 Formalization of protocol instantiations

As the title of this section suggests, we make a distinction between a protocol and a "protocol instantiation". We deem this a necessary distinction because protocols in the literature are often under-specified. A protocol may be implemented in any of several ways by the applications programmer.

The theorem prover Otter we used for protocol analysis is based on first order logic. Thus for formalizing a protocol, the environment properties and possible state transitions we use first order functions, predicates, implications and the usual connectives. At any step of a protocol run, predicates like

$state(xevents, xEknows, \ldots, xwho, xto, xmess, \ldots)$

[1] We note, however, that leaving this part of our model open allows for a very general tool. This is in part responsible for the success of the tool, as documented in later sections.

describe a state of the system: each predicate contains the list $xevents$ of actions already performed, the list $xEknows$ of data known by E, and additionally, for each agent and each run it assumes to be taking part in, components $xagent, xrun, xstep, xrandom, xkey$, etc. that describe the states of the agent in the particular run (e.g. data like session keys, nonces, etc., the agent has stored so far for later use, and the next step it will perform). In general the predicates can include as many agents' states and protocol runs as desired, but so far we have restricted this information to at most two agents, a server and the attacker in the context of two interleaving protocol runs. Furthermore, the predicates contain, among other data used to direct the search, the agent $xwho$ which sent the current message, the intended recipient xto and the message $xmess$ sent. Messages in our model are abstract objects, ordered tupels $(m_1, m_2, \ldots, m_r)$ of concatenated messages m_i which are viewed at an abstract level where the agents can distinguish between different parts.

Using these predicates we write formulas which describe possible state transitions. For the first step of the Needham-Schroeder Protocol (where agent A sends to the key server S the message A, B, R_A, see section 4), the action of S when receiving the message can be formalized as follows:

$$
\begin{aligned}
&send(xevents, xEknows, \ldots, xwho, xto, xmess, \ldots)\\
&\wedge\ xto = S\\
&\wedge\, xwho = E\\
&\wedge\ is\_agent(el(1, xmess)) \wedge\ is\_agent(el(2, xmess))\\
&\rightarrow\\
&send([\ldots|xevents], xEknows, \ldots, S, el(1, xmess),\\
&[scrypt(key(el(1, xmess), S), [el(3, xmess), el(2, xmess), new\_key,\\
&scrypt(key(el(2, xmess), S), [new\_key, el(1, xmess)])])], \ldots)
\end{aligned}
$$

where $[\ldots|xevents]$ includes the current send action, $el(k, xmess)$ returns the kth block of the message $xmess$ just received, and new_key stands for a number uniquely generated for each protocol run. By checking that the message was sent by E ($xwho = E$) we model the fact that agents only receive messages sent by E.

Whenever a message is intercepted by E (indicated by a formula $send(\ldots) \wedge xwho \neq E$), an internal procedure adds all that E can learn from the message to the knowledge list $xEknows$. The result of this procedure is a predicate $added\_all\_knows(\ldots)$. Implications $added\_all\_knows(\ldots) \rightarrow send(\ldots, xwho, \ldots, message1, \ldots) \wedge send(\ldots, xwho, \ldots, message2, \ldots) \wedge \ldots$ with $xwho = E$ formalize that after having intercepted a particular message, E has several possibilities to proceed, i.e. each of the predicates $send(\ldots, message, \ldots)$ constitutes a possible continuation of the protocol run. The messages E sends to the agents are constructed using the basic data in $xEknows$ and the specification of E's knowledge as well as some restrictions as discussed in the previous section.

Symmetric keys shared by agents X and Y are denoted by $key(X, Y)$, $key(X, pub)$ denotes the public key and $key(X, priv)$ the private key of agent X. For being able (if necessary) to distinguish between different cryptographic algorithms, we

use $scrypt(k, m)$ to formalise the encryption of message m with key k and the symmetric algorithm $scrypt$, $sdecrypt(k, m)$ formalizes the decryption of message m. Similarly an asymmetric algorithm being used is indicated by $acrypt(k, m)$ and a digital signature by $sig(k, m)$.

We use equations to describe the properties of the encryption and decryption functions, keys, etc., used in the protocol. For example, the equation

$$sdecrypt(x, scrypt(x, y)) = y$$

formalizes symmetric encryption and decryption functions where every key is its own inverse. These equations are used as demodulators in the deduction process.

The above formulas constitute the set of axioms describing the particular protocol instantiation to be analyzed. An additional set of formulas, the so-called "set of support", includes formulas that are specific for a particular analysis. In particular, we use a predicate describing the initial state of a protocol run (the agent that starts the run, with whom it wants to communicate, the initial knowledge of the intruder E, etc.) and a formula describing a security goal. An example of such a security goal is "if agent A believes it shares a session key with agent B, then this key is not known by E", formalized by $-(state(\ldots) \wedge xAstep = finished \wedge xAagent = B \wedge in\_list(xAkey, xEknows))$

Using these two formula sets and logical inference rules (specifically, we make extensive use of hyper resolution), Otter derives new valid formulas which correspond to possible states of a protocol run. The process stops if either no more formulas can be deduced or Otter finds a contradiction which involves the formula describing the security goal. When having proved a contradiction, Otter lists all formulas involved in the proof. An attack on the protocol can be easily deduced from this list.

Otter has a number of facilities for directing the search, for a detailed description see [30].

4 Known and new attacks

We used Otter to analyze a number of protocols. As an example, we show the analysis of the well-known symmetric Needham-Schroeder key distribution protocol [19]. The protocol comprises the following steps:

1. $A \longrightarrow S : A, B, R_A$
2. $S \longrightarrow A : \{R_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $A \longrightarrow B : \{K_{AB}, A\}_{K_{BS}}$
4. $B \longrightarrow A : \{R_B\}_{K_{AB}}$
5. $A \longrightarrow B : \{R_B - 1\}_{K_{AB}}$

S denotes a trusted key distribution server, $\{m\}_{K_{XY}}$ denotes the encryption of the message m using key K_{XY} (shared by agents X and Y), and R_X denotes a random number generated by agent X for a protocol run.

Whenever we model a protocol we are confronted with the problem that usually the checks performed by the parties upon receiving the messages are not

specified completely. This means that we must place ourselves in the situation of the implementer and simply encode the obvious checks. For example, consider step 3 of the previous protocol.

Here is a case where we have to decide what B's checks are. We did the obvious: (a) B decrypts using K_{BS}; (b)B checks that the second component of the decrypted message is some valid agent's name; and (c) B assumes, temporarily, that the first component of the decrypted message is the key to be shared with this agent.

However, Otter determined that this was a fatal flaw in the implementation. We call the type of attack found the "arity" attack, since it depends on an agent not checking the number of message blocks embedded in a cyphertext. The arity attack found by Otter works as follows: in the first message E impersonates B (denoted by E_B) and sends to S:

1. $E_B \longrightarrow S : B, A, R_E$

Then, S answers according to the protocol description and sends the following message:

2. $S \longrightarrow B : \{R_E, A, K_{AB}, \{K_{AB}, B\}_{K_{AS}}\}_{K_{BS}}$

The message is intercepted by E and then passed on to B, and B interprets it as being the third message of a new protocol run. Since this message passes the checks described above, B takes R_E to be a new session key to be shared with A and the last two prototol steps are as follows:

4. $B \longrightarrow E_A : \{R_B\}_{R_E}$
5. $E_A \longrightarrow B : \{R_B - 1\}_{R_E}$

where the last message can easily be generated by E since it knows its random number sent in the first protocol step (see [10] for a formalization of B's actions).

It is interesting to note that an applications programmer is quite likely to not include an "arity-check" in the code. This is because modular programming naturally leads to the coding of routines of the type

$$get(cyphertext, key, i)$$

which decodes *cyphertext* with *key* and returns the i^{th} element of the plaintext.

By including the arity check in B's actions when receiving message 3, we analyzed a different instantiation of the same protocol. This time Otter found the well-known Denning-Sacco attack [7], where the third message of an old protocol run is replayed to B and the attack run ends with B and the intruder E sharing an old session key that possibly was broken by E.

It seems that many protocols may be subject to the "arity" attack: our methods found that this attack is also possible on the Otway-Rees protocol (see [10] for a detailed description of the protocol and its analysis) and on the Yahalom protocol, as described in [6].

Among the known attacks which were also found by our methods are the Lowe ([13]) and the Meadows [17] attacks on the Needham-Schroeder public key protocol and the Simmons attack on the TMN protocol (see [29]). For the latter we relaxed the assumption of perfect encryption and modeled the homomorphic property of cryptoalgorithms like RSA [24] that allow to generate a ciphertext by multiplication.

Furthermore, we analyzed an authentication protocol for smartcards with a digital signature application standardized by the German standardization organisation DIN ([8]) and found the particular version of the standard to be subject to some interpretations which would make it insecure.

5 The extended model and its formalization

As already pointed out in the first section, e-commerce protocols are much more complex than traditional cryptographic protocols with respect to security requirements and threats. While in traditional protocols all agents are assumed to act correctly and the only threat is being posed by a hostile environment, in e-commerce protocols we have a situation where agents do not necessarily trust each other. A customer in a payment system might want to gain advantage of the system by not paying the agreed amount of money, a service provider may try to get away with not sending the service. So it is not so much the abilities of an outside intruder we are interested in but the abilities of the agents themselves that are taking part in a protocol.

As to security properties the protocols shall provide, we still have to deal with confidentiality and authenticity, but furthermore there are new properties that pose different questions. So besides the fact that protocol flaws may arise because of type confusion, we are interested in questions like "Is it possible for the client to receive the service without ever delivering the payment?", in which case we search for a system state where the service provider did deliver the service, the client owns whatever the protocol provides as a proof of having paid, but never did.

It is clear that the difference in security threats between classical and e-commerce protocols induces differences in the formalization of protocol and attack scenario. For the rest of the paper, we will refer to the analysis model for classical cryptoprotocols as the "classical model" and to that for e-commerce protocols as to the "e-commerce model".

Our communication and attack model for the analysis of e-commerce protocols is not the traditional Dolev-Yao model, where agents are assumed to act honestly and where an intruder is in the center of communication, intercepts every message and can send whatever message it is able to generate. Our model does contain an intruder, but additionally we assume that any number of agents participating in the system can be dishonest at any time. Regarding this, our model is more flexible than the model of mobile adversary introduced in [20], since here it is assumed that at most k out of n agents of the system can be corrupted. The capability of agents to generate faked messages is based on their knowledge, which is derived along the "rules" given in section 2. Furthermore, we also allow a situation where dishonest agents are capable of intercepting messages as well. This may or may not be relevant, depending on the environment in which to use the protocol.

In general our model allows an unrestricted number of dishonest agents, but so far we have restricted analysis runs to at most two. In the case of two dishon-

est agents, we assume that they cooperate, i.e. exchange whatever knowledge they have, except that they do not make available to each other their private keys. It is very unlikely that one analysis run with two dishonest agents acting independently gives more results than two analysis runs with one dishonest agent each.

As in the classical model, we are not concerned with the security of the algorithms that are being used in the protocols, i.e. we again assume perfect encryption. But in contrast to the analysis of classical protocols where we assume that agents are not able to distinguish between different types, in the e-commerce model we add types to the messages. Thus the agents can be modeled as being able or not being able to distinguish between different types of messages. This allows the analysis to concentrate on issues specific to e-commerce protocols rather than searching for protocol flaws caused by type confusion, but it keeps the search for type confusion based flaws possible.

Consequently, in the analysis runs performed so far we allowed dishonest agents only to send messages in the correct format, e.g. a message identification number will not be sent as a price, a signature will not be used as a random number, etc.

For the formalization of e-commerce protocols we had to make a few changes to predicates and formulas, the most important ones are listed below.

- The predicates additionally contain a list of agents that act dishonestly and a list of agents that cooperate (which allows for future extension of the number of cooperating agents). Furthermore, agents own a list of keys and a list for keeping information they extracted from messages not intended for them.
- In the classical cryptoprotocols we analysed so far there was no need to model an agent sending two messages subsequently. However, in e-commerce protocols the situation may be different, in particular when allowing for cooperating malicious agents whose actions are not fixed by the protocol description . Thus we introduced a "dummy message": If agent X sends first a message to agent Y and then a message to agent Z, we model this by having Y answer to X with $dummy$, after which X can proceed with sending the second message to Z.
- We model the interception of messages by dishonest agents principally in the same way as in the classical model: Whenever a message is sent by an honest agent $(send(\ldots) \wedge -in\_list(xwho, xbadguys))$, an internal procedure adds all that the dishonest agents can learn from the message to the respective knowledge list $xagentknows$. If there are cooperating agents, their knowledge lists are exchanged. Finally for each of the dishonest agents the messages to be sent are constructed, on the basis of the agent's knowledge and keys he owns. The messages are then sent to those agents that will accept the message with respect to the format (e.g. an order will not be send to a client, but only to a service provider). If we want to analyse a protocol in an environment where agents can not intercept messages, we skip the internal procedure of knowledge extraction and connect the reception of messages

directly with the generation of faked messages by way of demodulators such as $in\_list(xto, xbadguys) \longrightarrow send(\ldots) = construct\_messages(\ldots)$.

6 The Internet Billing Server Protocol

Using the above described model, we analyzed the Internet Billing Server Protocol (IBS protocol) developed by Carnegie Mellon University and described in [21]. Actually, [21] includes several different versions of protocol templates where the precise format of messages is left open. We chose the version that uses only asymmetric algorithms and made those changes we deemed necessary for a reasonable analysis.

The protocols were designed to enable sales of goods to be delivered over the network. There are three different types of agents: service providers, users (those agents which buy the goods) and the Billing Server, trusted by all agents, which handles the money transfer. Both service provider SP and user $User$ register with the billing server BS, which means in particular that accounts are opened and key pairs are issued by BS.

The protocol assumes that a signature algorithm with message recovery is used (an algorithm that allows to decipher a signature in order to get the plaintext again), $\{message\}_{SK_X}$ denotes the message signed with the private key of agent X. In the following, we describe our version of the IBS protocol. It consists of two phases, the first of which is price delivery, where the user asks some service provider for a price and the service provider answers with a price.

1. $User \longrightarrow SP \quad : \{ID, \text{request}, servicename\}_{SK_{User}}$
2. $SP \quad \longrightarrow User : \{ID, price\}_{SK_{SP}}$

The second phase is the service provision and payment phase:

3. $User \longrightarrow SP \quad : \{\{ID, price\}_{SK_{SP}}, ID, price\}_{SK_{User}}$
4. $SP \quad \longrightarrow BS \quad : \{\{\{ID, price\}_{SK_{SP}}, ID, price\}_{SK_{User}}\}_{SK_{SP}}$
5. $BS \quad \longrightarrow SP \quad : \{ID, \text{authorization}\}_{SK_{BS}}$
6. $SP \quad \longrightarrow User : \{ID, service\}_{SK_{SP}}$
7. $User \longrightarrow SP \quad : \{ID, \text{ackn}\}_{SK_{User}}$
8. $SP \quad \longrightarrow log \quad : \{\{ID, \text{ackn}\}_{SK_{User}}\}_{SK_{SP}}$
9. $SP \quad \longrightarrow BS \quad : log$

First, in step 3, the user orders the service. In step 4 the service provider asks the Billing Server for authorization of this service, which means in particular that the Billing Server checks that the user's account provides enough money. If so, the Billing Server transfers the price for the service from the user's to the service provider's account and keeps a record with all data relevant for this transaction. Then he authorizes the service (step 5) and the service is delivered (step 6). The user's last action is to send a message acknowledging that he received the service. The service provider collects all acknowledgement messages in a log file which is sent to the Billing Server in off-peak hours (step 9). When receiving the

log file, the Billing Server checks that the acknowledgement messages in the log file match the previously performed money transfers.

We added a message ID to all messages of the original version, and the constant "*request*" and the variable *servicename* to the first message, taking into account that a price will depend on the time it was given, the user it was given to, the service, and the service provider, that an authorization refers to a specific amount of money to be paid by a specific user to a specific service provider for a specific service, etc. The identification number both serves as a time stamp (it is issued by the user and unique for each run) and connects the messages of one run.

Some general assumptions where made in [21]:

1. In case of dispute, the user is always right. In consequence, if the service provider tries to cheat and the user complains about the charge, the service provider will always loose. On the other hand, the service provider can protect himself against malicious users by refusing access to the system.
2. All agents have access to a verifiable source of public keys, none of the keys gets compromised.
3. To secure the protocol against replay attacks etc., "time stamps, nonces, and other well documented means" can be used.

A number of security requirements the protocol imposes are explained in [21], some of which we list below.

1. The service provider can only be held responsible for those prices he indeed offered.
2. The user must be sure he will only be charged for services he received for the price mutually agreed on.
3. The service provider must be sure that the user is not able to receive a service and deny the receipt.
   Clearly, the protocol does allow the user to deny having ever received the service: he can just refuse to send the acknowledgement message. According to [21], the protocol is designed for services with small value only, thus a service provider can always deny a user that has refused to send acknowledgement messages further access to the system.

6.1 Our assumptions and formalization

To model this particular version of the IBS protocol, we used the following assumptions:

1. The system consists of two Users $User1, User2$, two service providers $SP1$, $SP2$, the Billing Server BS and an outside intruder E. The Billing Server is the only agent that always behaves according to the protocol description. Furthermore, we have two services $service1, service2$ with respective service names and different prices to start with for the service providers. The price is incremented after having been used. (However, we can use a initial state where both service providers use the same price as a starting point.)

2. All agents own valid key pairs and the public key of all other agents.
3. All predicates contain, for each user and each service provider, an additional component *xagentmemory*. For each protocol run an agent is engaged in we have an entry in this component that contains the ID, the user / service provider he is communicating with, the service that is being negotiated, the price given and the state the agent is in. Equivalently, the Billing Server holds a list of authorization requests (*xauthrequ*), each of which contains ID, *price*, $User$ and SP being engaged in the particular negotiation. Additionally, BS holds a list with the accounts of users and service providers.
4. Although in [21] it is not explicitly said so, we assume that all signatures are checked. Since the protocol does not provide information on who signed the messages, we added a further component *xdetmessages* to the predicates that includes the ID, the message type (i.e. *message*1, *message*2, etc.) and the signers of the message, starting with the outermost signature, the next inner one, then the innermost signature (if there are that many). This allows to identify the signer of the first message (a service provider can not know who has sent a particular price request), and to identify which run the message belongs to by use of the ID.
   Whenever in the recipient's *xagentmemory* there exists already an entry with the ID given in *xdetmessages*, the public keys to be used for signature verification are determined by using the public keys of the user and service provider, respectively, given in this particular entry. This models the fact that a user will only accept a service provided by the service provider he ordered the service from, that a service provider will only accept an order that was sent by the user he offered the specific price to, etc. However, there may be situations where only the integrity of the message needs to be ensured and the actual signer is not of interest. For example, a user might not be interested in knowing who actually sent the mpeg file he ordered. To model this we can determine the public keys to be used for signature verification by using the agents given in *xdetmessages*.

7 Our analysis of the IBS protocol

As already pointed out in the previous section, there are a number of security requirements to be met. In the following we will list a few of these requirements and the formula that models the respective property (thus the formula to be found by Otter for finding a contradiction and therefore an attack).

1. The service provider can only be held responsible for those prices he indeed offered. This can be formalized with the formula $-(state(\ldots) \wedge give\_price(xSP, xID) > give\_price(xauthrequ, xID))$.
2. The user must be sure he will only be charged for services he received at the price mutually agreed on. These are actually two requirements that can be captured by $-(state(\ldots) \wedge -service\_delivery(xauthrequ, xevents))$ and $-(state(\ldots) \wedge give\_price(xUser, xID) > give\_price(xauthrequ, xID))$.

3. The service provider must be sure that the user is not able to receive a service and deny the receipt: $-(state(\ldots) \wedge denial\_ackn(xUsermemory, xSPmemo-ry))$.

The values of the predicates $give\_price, denial\_ackn$, etc., are determined by means of conditional demodulation.

For the analysis, in general we assumed that all possible checks are performed. This includes that the service provider checks, receiving message 3, that the price given in his own signature (which is part of the message signed by the user) is the same as the one signed by the user; that the Billing Server checks that the two signatures of the service provider in the request for authorization (inner and outer) are performed by the same service provider, etc. This also includes that the Billing Server checks, when receiving an authorization request, that he has not yet received a request with the ID given.

The first protocol flaw we found is the one already stated in [21]: By modeling a malicious user that can stop the protocol run at any given point, we found (which is not surprising) a state in which the user has stopped the run and the service provider has not received an acknowledgement message.

We then disabled the user's ability to stop runs and let the analysis run with a malicious user and a malicious service provider. Now Otter found that the protocol can be run without performing step 6, i.e. without service delivery, and the Billing Server still accepts the log file containing the user's acknowledgement message. Thus the protocol does allow a situation where the user is charged without having received a service. One may argue that this can not be considered an attack, since it is only possible with the active cooperation of the user, and thus is no violation of the security requirement that a user may only be charged for services actually received. In many cases this will be right. However, our analysis reveales the fact that the messages received by the Billing Server do not constitute a proof that a service was delivered but a proof that the user agrees on being charged a specific amount of money. There may be environments where it matters whether or not the Billing Server can be used for transfering money from one account to another without a service being involved, environments where any kind of "money laundrey" has to be avoided.

Our analysis also shows that care must be taken when specifying the security requirements. One way to formalize security requirements precisely is by way of system states that describe a violation of the desired property. In an environment where the above described protocol run is unacceptable, the formula $-(state(\ldots) \wedge -service\_delivery(xauthrequ, xevents))$ captures the fact that **no** user shall be charged without having received a service. If "money laundrey" is of no importance, the requirement can be relaxed by adding $\wedge -in\_list(xuser, xbadguys)$.

8 Conclusions

As is well known, there does not exist an algorithm which can infallibly decide whether a theorem is true or not. Thus the use of Otter, and per force of any

theorem proving tool, is an art as well as a science. The learning curve for using Otter is quite long, which is a disadvantage. Furthermore, one can easily write Otter input which would simply take too long to find a protocol failure even if one exists and is in the search path of Otter's heuristic. Thus, using this kind of tool involves the usual decisions regarding the search path to be followed.

Nevertheless, our approach has shown itself to be a powerful and flexible way of analyzing protocols. By making all of the agents' actions explicit, our methods reveal implicit assumptions that are not met by the protocol being analyzed. In particular, our methods found protocol weaknesses not previously found by other formal methods. On the other hand, as we are concerned with protocol validation rather than verification, if our analysis does not find an attack, we can not conclude that the protocol is secure in general. All we know is that under certain assumptions concerning the security properties of the cryptoalgorithms used and the abilities of malicious agents, a certain state is unreachable.

Furthermore, we have successfully adapted our methods to address the specific needs of e-commerce protocol analysis. The new model for the analysis of e-commerce protocols is flexible enough to be used for different threat scenarios with respect to possibilities of malicious agents. Our analysis of a specific version of the IBS protocol both shows our methods to be applicable to e-commerce protocols and emphazises that care must be taken when identifying the security requirements of such a protocol. A way to formalize the desired properties is to use formulas that describe a system state where the property in question is violated. Future work will include the formalization of more security properties relevant for e-commerce protocols and the application of our methods to other e-commerce protocols.

References

1. G. Bella and L.C. Paulson. Kerberos version iv: Inductive analysis of the secrecy goals. In *5th European Symposium on Research in Computer Security*, Lecture Notes in Computer Science, pages 361–375. Springer-Verlag, 1998.
2. M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. In *Annual Symposium on the Theory of Computing*. ACM, 1998.
3. M. Bellare and P. Rogaway. Provably secure session key distribution - the three party case. In *Annual Symposium on the Theory of Computing*, pages 57–66. ACM, 1995.
4. R. Berger, S. Kannan, and R. Peralta. A framework for the study of cryptographic protocols. In *Advances in Cryptology – CRYPTO '95*, Lecture Notes in Computer Science, pages 87–103. Springer-Verlag, 1985.
5. S. Brackin. Automatically detecting authentication limitations in commercial security protocols. In *Proc. of the 22nd National Conference on Information Systems Security*, October 1999.
6. M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. Report 39, Digital Systems Research Center, Palo Alto, California, Feb 1989.
7. D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24:533–536, 1982.

8. DIN NI-17.4. *Spezifikation der Schnittstelle zu Chipkarten mit Digitaler Signatur-Anwendung / Funktion nach SigG und SigV, Version 1.0 (Draft)*, November 1998.
9. D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.
10. S. Gürgens and R. Peralta. Efficient Automated Testing of Cryptographic Protocols. Technical report, GMD German National Research Center for Information Technology, 1998.
11. N. Heintze and J.D. Tygar. A Model for Secure Protocols and their Compositions. In *1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 2–13. IEEE Computer Society Press, May 1994.
12. R. Kailar. Accountability in Electronic Commerce Protocols. In *IEEE Transactions on Software Engineering*, volume 22, pages 313–328. IEEE, 1996.
13. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In *Second International Workshop, TACAS '96*, volume 1055 of *LNCS*, pages 147–166. SV, 1996.
14. W. Marrero, E. M. Clarke, and S. Jha. A Model Checker for Authentication Protocols. In *DIMACS Workshop on Cryptographic Protocol Design and Verification*, http://dimacs.rutgers.edu/Workshops/Security/, 1997.
15. C. Meadows. A system for the specification and verification of key management protocols. In *IEEE Symposium on Security and Privacy*, pages 182–195. IEEE Computer Society Press, New York, 1991.
16. C. Meadows. Formal Verification of Cryptographic Protocols: A Survey. In *Advances in Cryptology - Asiacrypt '94*, volume 917 of *LNCS*, pages 133 – 150. SV, 1995.
17. C. Meadows. Analyzing the Needham-Schroeder Public Key Protocol: A Comparison of Two Approaches. In *Proceedings of ESORICS*, Naval Research Laboratory, 1996. Springer.
18. C. Meadows and P. Syverson. A formal specification of requirements for payment transactions in the SET protocol. In *Proceedings of Financial Cryptography*, 1998.
19. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, pages 993–999, 1978.
20. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proceedings of PODC*, pages 51–59, 1991.
21. K. O'Toole. The Internet Billing Server - Transaction Protocol Alternatives. Technical Report INI TR 1994-1, Carnegie Mellon University, Information Networking Institute, 1994.
22. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
23. L. C. Paulson. Inductive Analysis of the Internet Protocol TLS. *ACM Trans. on Information and System Security*, 2(3):332–351, 1999.
24. R. L. Rivest, A. Shamir, and L. A. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
25. B. Roscoe, P. Ryan, S. Schneider, M. Goldsmith, and G. Lowe. *The modelling and Analysis of Security Protocols.* Addison Wesley, 2000.
26. C. Rudolph. A Formal Model for Systematic Design of Key Establishment Protocols. In *Information Security and Privacy (ACISP 98)*, Lecture Notes in Computer Science. Springer Verlag, 1998.
27. S. Schneider. Verifying authentication protocols with CSP. In *IEEE Computer Security Foundations Workshop.* IEEE, 1997.

28. V. Shoup and A. Rubin. Session key distribution using smart card. In *Advances in Cryptology - EUROCRYPT '96*, volume 1070 of *LNCS*, pages 321–331. SV, 1996.
29. M. Tatebayashi, N. Matsuzaki, and D. Newman. Key Distribution Protocol for Digital Mobile Communication Systems. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89*, volume 435 of *LNCS*, pages 324–333. SV, 1991.
30. L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning - Introduction and Applications*. McGraw-Hill, Inc., 1992.