Verifying Second-Level Security Protocols

Giampaolo Bella[1,2] Cristiano Longo[2] Lawrence C Paulson[1]

[1] Computer Laboratory, University of Cambridge
15 JJ Thomson Avenue, Cambridge CB3 0FD (UK)
{gb221,lcp}@cl.cam.ac.uk

[2] Dipartimento di Matematica e Informatica, Università di Catania
Viale A. Doria 6, I-95125 Catania (ITALY)
{giamp,longo}@dmi.unict.it

Abstract. A *second-level security protocol* is defined as a security protocol that relies on an underlying security protocol in order to achieve its goals. The verification of classical authentication protocols has become routine, but second-level protocols raise new challenges. These include the formalisation of appeals to the underlying protocols, the modification of the threat model, and the formalisation of the novel goals. These challenges have been met using Isabelle and the Inductive Approach [14]. The outcomes are demonstrated on a recent protocol for certified e-mail delivery by Abadi et al. [2].

1 Introduction

The development of *security protocols* during the last two and a half decades has mainly focused on the fundamental goals of *authentication*, which confirms the identity of a peer, and of *confidentiality*, which confirms that a message is kept from attackers. Somewhat simplistically, security protocols have often been referred to as *authentication protocols* [9, 10]. However, the growth of Internet transactions and distributed computing in general require more sophisticated goals such as *anonymity*, *non-repudiation* [18], *delegation* [4] and *certified delivery* [2]. No one can predict what security goals will become important next year.

Many recent protocols rely on some other protocol as a primitive. For example, the fair-exchange protocol by Asokan et al. [3] presupposes that the peers authenticate each other by means of an authentication protocol before they engage in a transaction whereby they commit to a contract. The certified e-mail protocol by Abadi et al. requires the establishment of a secure channel between the e-mail recipient and the trusted third party: "in practice, such a channel might be an SSL connection" [2]. But SSL is already rather complicated [15], hence a protocol that employs it becomes even more complicated as a whole, unless we assume that SSL works correctly. These observations inspire our definition of *second-level* protocols.

Definition 1 (Second-Level Protocol). *A* second-level security protocol *is a security protocol that relies on the goals of underlying authentication protocols in order to achieve its goals.*

This concept is natural. Any kind of construction customarily makes use of existing constructions, and this process can be iterated. Likewise, our definition can be trivially generalised for n^{th}*-level* protocols, which employ $(n-1)^{st}$*-level* protocols. In turn, classical authentication protocols can be seen as *first-level* protocols, which adopt 0^{th}*-level* protocols, the transfer protocols.

The verification of authentication protocols can be considered mature. Innumerable approaches have been taken [8, 11, 12, 14, 16]; a wide variety of protocols can be formalised; the verification is often highly or fully automated. However, second-level protocols raise new challenges for formal verification. We briefly outline them here.

Underlying goals. It is not obvious how to treat the goals made available by the underlying authentication protocols. Should we examine the interaction of authentication and second-level protocols directly, combining them to form one giant protocol? Surely not. Should we instead take for granted the authentication protocols' goals during the verification of the second-level protocols? By following well-established principles of hierarchical verification, we settled for the latter course. The former is conceptually simpler and avoids the danger of missing a low-level interaction between the two levels, but it becomes infeasible as more protocols are combined.

Threat model. Dolev-Yao's threat model is the standard for authentication protocols. It consists of a single attacker who monitors the entire network, reuses the intercepted messages as he pleases, and yet cannot break ciphertexts. But what is the threat model for second-level protocols? Our choice was to limit Dolev-Yao's model as follows. If a second-level protocol rests on some authenticated communication, then the attacker must be prevented from interposing. If that communication is confidential, then the attacker must be prevented from overhearing.

New goals. If we aim at verifying new goals, we must first solve the problem of their formalisation. New security requirements will be increasingly complex and difficult to formalise. Concentrating on second-level protocols, we have found simple formalisations for the goals of certified e-mail delivery, and, most importantly, simple strategies for proving them. They will be presented below. Goals of other second-level protocols may require dedicated treatment.

This paper addresses the challenges given above. We start with the Inductive Approach [14] to protocol verification and make it capable of verifying second-level protocols. Our modifications, described in Sect. 2, are surprisingly simple. The currently formalised network events can be quickly adapted to the new protocols. Then, Sect. 3 describes the certified e-mail protocol developed by Abadi et al. [2]. Sect. 4 introduces our inductive model for the protocol, and Sect. 5 describes our findings. Sect. 6 concludes the paper.

There is little work that is related to ours. Abadi and Blanchet are publishing their verification of the same protocol [1] using Blanchet's verifier [8]. Their study is extremely detailed and mostly automatic, human intervention being required only occasionally. For example, "The fact that messages reach their destination

(. . .) cannot be taken into account by our verifier, so it cannot prove the required properties in a fully automatic way" [1, p.15]. Both our works appear to confirm that the protocol meets its goals, though a deeper comparison is necessary.

Shmatikov and Mitchell have used model-checking techniques to analyse what we would call a second-level protocol [17]. Their model is kept small to make exhaustive search feasible, and appears to be precisely tailored to the specific protocol under consideration. By contrast, our aim below is to provide a general treatment of second-level protocol verification. Because we use theorem proving rather than model checking, we do not have to limit the size of the model.

2 Modelling and Verifying Second-Level Protocols

Our approach to analysing second-level security protocols is built upon our existing work on analysing authentication protocols [5, 14] using inductive definitions. The machinery is mechanised in Isabelle/HOL, so modelling and verifying a protocol in fact involves interaction with that theorem prover.

We outline the original approach (§2.1) and describe how we extended it to cope with second-level protocols (§2.2).

2.1 The Inductive Approach

Our approach is to formalise the system inductively [14]. This defines an operational semantics that has much in common with Lowe's CSP formalisations [13], except that the models are infinite. The participants include the honest ones, who execute the protocol faithfully; one of these is designated as a trusted third party. There is a spy, who eavesdrops on network traffic and might send any messages he is able to generate. We only verify safety properties: for example, we cannot reason about denial of service. Proving a protocol guarantee involves demonstrating that it holds in all execution traces, no matter what the spy does.

Event histories are lists (built in reverse order) containing three kinds of elements.

- The event $\mathsf{Says}\, A\, B\, X$ means that A has sent message X to B. However, sending does not imply reception.
- The event $\mathsf{Gets}\, B\, X$ means that B has received message X from the network. As we normally consider the channel to be unauthenticated, B does not know who sent the message.
- The event $\mathsf{Notes}\, A\, X$ means that A has stored message X internally. It represents a local state change.

Messages are a recursive datatype that includes the following constructors:

- $\mathsf{Agent}\, A$ denotes the name of the agent A.
- $\mathsf{Number}\, N$ denotes a guessable number, where N is a non-negative integer.
- $\mathsf{Nonce}\, N$ is like $\mathsf{Number}\, N$ but the number is treated as non-guessable; it models long, randomly generated byte strings.

– Key K denotes a key and is treated as non-guessable.
– Hash X denotes the cryptographic hash of the message X.
– Crypt $K\,X$ denotes encryption of message X with K.
– $\{\!|X_1, \ldots, X_n|\!\}$ denotes the concatenation of the messages $X_1, \ldots, X_n$.

Protocol definitions and proofs involve several additional functions.

– used *evs* denotes the set of all message components that appear in the trace *evs*, so Nonce $N \notin$ used *evs* expresses that N is a fresh nonce.
– parts H denotes the set of all message components (including the plaintexts of encryptions) that appear in the set of messages H.
– analz H is a subset of parts H, namely the components that are effectively derivable from the set of messages H. Plaintexts of encryptions are derivable provided the corresponding key is also derivable.
– synth H denotes the set of messages that can be built up using elements of H and guessable values.
– priEK A and priSK A denote the private keys (encryption and signature) of the agent A.
– pubEK A and pubSK A denote the public keys (encryption and signature) of the agent A.
– symKeys denotes the set of all symmetric keys; the complement of this set denotes the asymmetric keys.
– bad denotes the set of compromised agents (see below).

1.	$A \longrightarrow B$	:	A, Na
2.	$B \longrightarrow A$	:	$\{\!|Na|\!\}_{Kb^{-1}}$

```
Nil:   "[] ∈ dap"

DAP1:  "⟦evs1 ∈ dap; Nonce Na ∉ used evs1⟧
        ⟹ Says A B {|Agent A, Nonce Na|} # evs1 ∈ dap"

DAP2:  "⟦evs2 ∈ dap; Gets B {|Agent A, Nonce Na|} ∈ set evs2⟧
        ⟹ Says B A (Crypt (priSK B) (Nonce Na)) # evs2 ∈ dap"

Recp:  "⟦evsr ∈ dap; Says A B X ∈ set evsr⟧
        ⟹ Gets B X # evsr ∈ dap"

Fake:  "⟦evsf ∈ dap; X ∈ synth(analz(knows Spy evsf))⟧
        ⟹ Says Spy B X # evsf ∈ dap"
```

Fig. 1. A demo authentication protocol (DAP) and its inductive model

Fig. 1 presents a trivial authentication protocol and its inductive definition. The latter consists of four introduction rules defining the constant dap, which denotes the set of traces permissible with this protocol. By rule *Nil*, the empty trace is permissible. Rule *DAP1* represents the first protocol step: A chooses a fresh nonce and sends it to B. Rule *DAP2* represents the second protocol step: if B receives a suitable message, he signs the nonce it contains using his private key and sends it to A. He doesn't know that the message came from A, but sends it to the principal named in the first component of the message he received. Rule *Recp* says that if a message is sent, it might be received. Rule *Fake* allows the spy to generate a message X using material gleaned from past traffic, and send it to anybody.

2.2 Adapting the Inductive Approach

The Inductive Approach had to be extended to face the three challenges sketched in the introduction, a process that required much thought. However, the extensions turned out to be technically simple and quick to implement.

Modelling the underlying protocols' goals. According to Definition 1, the underlying protocols are authentication protocols. Therefore, the main goals that we need to model are authentication and confidentiality. Other, minor, goals may also be of interest.

Authentication. The sender identity A of a $\mathsf{Says}\, A\, B\, X$ event cannot be altered in the model once that event occurs on a trace. When we formalise an authentication protocol, we must not allow an agent to inspect the sender identity; the originator of a message remains unknown unless it is conveyed by the message itself. We usually formalise message reception using the Gets event, which does not mention the sender. In earlier work [14], we formalised message reception using the event $\mathsf{Says}\, A'\, B\, X$, taking care to ensure that the value of A' (the true sender) was never used.

On the basis of these observations, one strategy to formalising authentication is to allow Says events among rule preconditions. For example, $\mathsf{Says}\, A\, B\, X$ would signify that B can authenticate X as coming from A. This is the right way to model an authenticated channel that does not offer confidentiality, because the spy can read X from the event $\mathsf{Says}\, A\, B\, X$.

Confidentiality. What if the channel must be confidential? We could extend our definitional framework, introducing an additional event $\mathsf{ConfSays}\, A\, B\, X$ for sending a message confidentially. This would require extending the definition of the function knows, which formalises each agent's knowledge. The new event $\mathsf{ConfSays}\, A\, B\, X$ would make X available only to the designated recipient, B. If we include the event $\mathsf{ConfSays}\, A\, B\, X$ as the precondition of a rule and allow other references to A, who is the true sender, then we have modelled an authenticated, confidential channel. If we forbid other references to A, then our confidential

channel is unauthenticated. We performed some experiments using the new event but abandoned them when we realised that the original definitional framework was already sufficient to model secure channels.

The Notes event formalises an agent's changing his internal state. It has the form $\mathsf{Notes}\, A\, X$, where X is a message (perhaps the result of a computation) being stored for future reference. We can formalise a confidential transmission of a message X from A to B by the event

$$\mathsf{Notes}\, A\, \{\!|A, B, X|\!\} \tag{1}$$

as demonstrated by rule `DSLP1` in Fig. 2. The figure shows a demo second-level protocol that differs from the demo authentication protocol seen above only in the first step, which now takes place over an authenticated, confidential channel. Event 1 must be included as precondition of a new inductive rule formalising reception of the confidential message, because reception is in general not guaranteed even on a confidential channel. The new reception rule must introduce the event

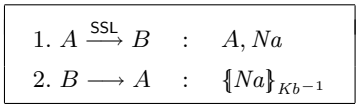
$$\mathsf{Notes}\, B\, \{\!|A, B, X|\!\} \tag{2}$$

signifying that B receives X confidentially, as demonstrated by rule `RecpN` in Fig. 2. Event 2 must be included as precondition of the rule formalising B's actions upon reception of the confidential message X. The message is therefore authenticated to arrive from the agent whose identity appears as first component of the noted message. This is demonstrated by rule `DSLP2` in Fig. 2.

No additional modification is necessary to the model of a second-level protocol, as highlighted by a comparison of Fig. 2 with Fig. 1. In particular, the `Fake` rule remains unvaried. However, some protocols require an additional rule, allowing the spy to send arbitrary messages on previously established confidential connections. Such a rule would insert an event of the form $\mathsf{Notes}\, \mathsf{Spy}\, \{\!|\mathsf{Spy}, B, X|\!\}$. Also, it is important to make sure that other uses of Notes do not involve messages beginning with two agent names, to avoid confusion with our representation of transmission along secure channels.

Guaranteed Delivery. Other minor goals of authentication protocols can be formalised using similar techniques. For example, distribution of a session key to a pair of agents can be formalised by an inductive rule that gives both agents Notes events containing a key, with a precondition that the key is fresh. Another goal is guaranteed delivery, which can be implemented by sending a message repeatedly until there is an acknowledgement. This goal can be easily formalised by introducing the event for receiving a message at the same time as the event for sending it. Even simpler is just to introduce the former event: the sender magically causes the message to reach the recipient. This formalisation will be vastly used below (§4) to model Abadi et al.'s certified e-mail protocol. With either approach to guaranteed delivery, no reception rule in the style of `RecpN` (Fig. 2) is necessary.

Notes events are affected by a detail of our model, the set bad of *compromised agents*. These are honest agents that have somehow come under the control of




```
Nil:    "[] ∈ dslp"

DSLP1:  "⟦evs1 ∈ dslp; Nonce Na ∉ used evs1⟧
         ⟹ Notes A {|Agent A, Agent B, Nonce Na|} # evs1 ∈ dslp"

DSLP2:  "⟦evs2 ∈ dslp;
          Notes B {|Agent A, Agent B, Nonce Na|} ∈ set evs2⟧
         ⟹ Says B A (Crypt (priSK B) (Nonce Na)) # evs2 ∈ dslp"

Recp:   "⟦evsr ∈ dslp; Says A B X ∈ set evsr⟧
         ⟹ Gets B X # evsr ∈ dslp"

RecpN:  "⟦evsr ∈ dslp; Notes A {|Agent A, Agent B, X|} ∈ set evsr⟧
         ⟹ Notes B {|Agent A, Agent B, X|} # evsr ∈ dslp"

Fake:   "⟦evsf ∈ dslp; X ∈ synth(analz(knows Spy evsf))⟧
         ⟹ Says Spy B X # evsf ∈ dslp"
```

Fig. 2. A demo second-level protocol (DSLP) and its inductive model

the spy, perhaps through a security lapse. The spy knows their private keys and can read their Notes. This detail is consistent with our use of Notes above, since we can expect the spy to grab anything that a compromised agent receives, even via a secure channel. The model does not constrain bad other than asserting that the spy is in this set.

Traditional authentication protocols are designed to protect honest agents from the rest of the world, and a typical guarantee will hold provided both peers are uncompromised. However, some protocols are intended to protect a principal even from its peer, about whom we make no assumptions. An agent A who executes a non-repudiation protocol or a certified e-mail protocol with B requires the protocol goals *especially* if B is bad.

Let us emphasise what we have provided here, namely a formalisation of the security properties assumed by a second-level protocol. We do not formalise the specific underlying authentication protocol that achieves those properties. Following a hierarchical verification strategy, it is irrelevant what is hidden inside the black box (whether SSL or Kerberos, for example) that is assumed to provide the goals. To our knowledge, this is the first time that principles of hierarchical verification have been spelled out in the context of security protocols.

Adapting the threat model. The threat model formalised in the Inductive Approach is a Dolev-Yao spy (§2.1). He monitors the network traffic by means of knows, can analyse that traffic by means of analz, and can synthesise new messages from the analysed components by means of synth. The spy can send a so formed message X to anyone, as formalised by rule `Fake` in Fig. 1, which introduces the event $\mathsf{Says}\,\mathsf{Spy}\,B\,X$ in the current trace, B being a free variable.

In principle, it is not obvious what is the threat model for second-level protocols, but we believe that they should still achieve their goals against a Dolev-Yao spy, as illustrated below on the certified e-mail protocol by Abadi et al. Therefore, we decide to maintain the existing spy in such a way that he cannot interfere with the goals of the underlying authentication protocols, though he can execute these protocols himself.

In other words, we analyse second-level protocols under a threat model that is the same as that for authentication protocols, except that it does nothing against those communications that are secured by authentication protocols. Modelling secure communications using Notes events, as explained above, yields this threat model for free — that is, with no changes to our definitional framework.

Modelling and verifying the novel goals. Abstractly, the main goal of certified e-mail delivery resembles a logical equivalence. Given an e-mail m, a sender S and an intended receiver R for m, in the assumption that S sends m to R, it must be the case that R receives m if and only if S obtains the receipt d that R received m.

Let evs be a generic trace of the protocol model;
let $\mathsf{Says}\,S\,R\,X$ *be an event in evs such that X features m;*
then

$$m \in \mathsf{analz}(\mathsf{knows}\,R\,evs) \iff d \in \mathsf{analz}(\mathsf{knows}\,S\,evs).$$

Fig. 3. Abstract formalisation of certified e-mail delivery

The Inductive Approach comes with enough operators to model this goal. Fig. 3 specifies certified e-mail delivery at a very high level. As it is customary, the goal is expressed over a generic trace of events *evs* of the inductive protocol model. The Says event in the preconditions expresses the role of sender played by S, that of receiver played by R, and features the e-mail m in a form that depends on the specific protocol, such as encrypted or hashed. The delivery receipt, d, is presumably built up from m, S and R. Then, the goal can be expressed as a logical equivalence, one implication for each protocol participant. Most importantly, the formalisation must express that agents who conform to the protocol be protected from agents who do not.

Implication ($\Longrightarrow$) reads as follows: if the receiver can derive the e-mail from the portion he sees of the network traffic on *evs*, then the sender can derive the

corresponding delivery receipt from the portion he sees of the network traffic on *evs*. This guarantee confirms that the sender never has a disadvantage over the receiver, even if the receiver is able to derive the e-mail off-line from the analysed components.

Implication ($\Longleftarrow$) reads as follows: if the sender can derive a delivery receipt, then the receiver can derive the corresponding e-mail. When the sender submits his delivery receipt to a judge, this guarantee counts as evidence against the receiver. We assume that a delivery receipt involves a digital signature or other mechanism to prevent its being forged; we could explicitly exclude this danger by formalising the precondition as $d \in \mathsf{synth}(\mathsf{analz}(\mathsf{knows}\, S\ evs))$, since the operator synth expresses message creation.

Given a specific protocol, the abstract formalisation of certified e-mail delivery needs to be refined. Despite its nice symmetry, its conclusions are weak because the operator analz represents a potentially unlimited series of decryptions by available keys. A stronger formalisation would replace the use of analz by a reference to a specific protocol message that delivers the required item. The refined Implication ($\Longrightarrow$) says that if the receiver can compute the e-mail by an unlimited amount of work, then the sender has been given the corresponding delivery receipt. The refined Implication ($\Longleftarrow$) says that if the sender can compute a delivery receipt by an unlimited amount of work, then the receiver has been given the corresponding e-mail. Note that we cannot express this refined formalisation unless we know the precise format of the protocol messages.

An additional problem is that, at the time of this writing, we lack techniques for reasoning about the knowledge of agents other than the spy. Specifically, we do not know how to prove anything useful from the assumption $X \in \mathsf{analz}(\mathsf{knows}\, A\ evs)$ unless $A = \mathsf{Spy}$. Sometimes, we can correct the situation by proving something stronger that no longer involves analz. Here is an improved version of Implication ($\Longleftarrow$):

> If a delivery receipt has been created at all, then the receiver has been given the corresponding e-mail.

This version does not refer to the sender. We have strengthened the guarantee while eliminating the need to reason about the sender's knowledge.

We cannot apply this technique to Implication ($\Longrightarrow$) because obviously the message m will have been created. Instead, we are forced to divide Implication ($\Longrightarrow$) into two separate properties. One concerns the spy and is proved by reasoning about $\mathsf{analz}(\mathsf{knows}\, \mathsf{Spy}\ evs)$, which we know how to do. The other concerns an honest agent, and states that if R is given the message (in the normal way) then S has been given the receipt.

Further refinements depend upon the specific delivery mechanism used in the protocol. The e-mail protocol we consider below uses a key to protect the message. Later, R is given this key, so that she can decrypt the message. We therefore refine the statements above, replacing "R has the e-mail" by "R has the key to the e-mail." We can also formalise a further guarantee, namely that the key never reaches the spy. This guarantee, which is of obvious value to both

parties, is formalised and proved much as it would be for a typical authentication protocol.

To prove these guarantees, we have been able to reuse the techniques previously developed on a non-repudiation protocol [7], which has the similar aim of protecting one agent from another.

3 Abadi et al.'s Certified Email Protocol

Abadi et al. [2] have recently designed a protocol for certified e-mail delivery that appears to have many practical advantages. Although it requires a trusted third party (TTP), this TTP is stateless and lightweight; it never has access to the clear-text of the transmitted messages. The burden on the TTP is independent of the message size. No public-key infrastructure is necessary. The TTP must have signature and encryption keys, but other principals merely share a secret with the TTP, such as a password.

The gist of the protocol is that the e-mail sender sends his e-mail encrypted under a symmetric key to the intended receiver. (This is similar in spirit to the previously mentioned non-repudiation protocol: send the encrypted message first, and the corresponding key only later.) Here, the sender also attaches the symmetric key encrypted under the TTP's public encryption key. Then, the recipient forwards the message to the TTP in order to obtain the symmetric key. (In the non-repudiation protocol, it is up to the sender to lodge the key with the TTP.) As the TTP releases the key, it also releases a certificate documenting the transaction to the sender.

Fig. 4 presents a protocol version, simplifying the headers and authentication options. The fullest authentication option, whereby the receiver should authenticate to both the sender and the TTP, is assumed. Our notation purposely makes no distinction between the various uses of symmetric encryption, asymmetric encryption, and digital signatures by using subscripts in all cases.

In step 1, the sender S sends the receiver R the encrypted e-mail $\{\!|m|\!\}_k$, a query q, and a certificate for the TTP, called $S2TTP$. The query is part of the authentication mechanism between R and S. They are required to agree beforehand on some acknowledgement function to link a query q to its response r. The certificate is encrypted under the TTP's public encryption key, ($\mathsf{pubEKTTP}$), and contains the symmetric key k that protects $\{\!|m|\!\}_k$ along with a hash linking $\{\!|m|\!\}_k$ to the query/response pair.

In step 2, R issues a response r' to the query q' just received, which is not necessarily the same as S's query because the network is insecure. Then, R uses his query/response pair along with the received ciphertext em' to build his own hash h_R. Finally, R wraps this up with the received certificate and with his password RPwd and sends the outcome to the TTP on secure channel. Presumably, R creates this channel by running the SSL protocol with authentication of TTP.

In step 3, the TTP verifies that the received certificate has the expected form by decrypting it by its private decryption key. Then, the TTP authenticates R by the password and verifies that h'_S found inside the ticket matches h'_R. A positive

Abbreviations

$$
\begin{array}{lcl}
h_S & = & \mathsf{Hash}(q, r, \{\!|m|\!\}_k) \\
h_R & = & \mathsf{Hash}(q', r', em') \\
S2TTP & = & \{\!|S, k, R, h_S|\!\}_{(\mathsf{pubEKTTP})}
\end{array}
$$

Steps

$$
\begin{array}{lrcl:l}
1. & S & \longrightarrow & R & \mathsf{TTP}, \{\!|m|\!\}_k, q, S2TTP \\
2. & R & \xrightarrow{\mathsf{SSL}} & \mathsf{TTP} & S2TTP', \mathsf{RPwd}, h_R \\
3. & \mathsf{TTP} & \xrightarrow{\mathsf{SSL}} & R & k', h'_R \\
4. & \mathsf{TTP} & \longrightarrow & S & \{\!|S2TTP''|\!\}_{(\mathsf{priSKTTP})}
\end{array}
$$

Fig. 4. Abadi et al.'s certified e-mail protocol

check signifies that S and R agree on the authentication mechanism. If satisfied, the TTP replies to R, delivering the key found inside the ticket. This reply goes along the secure channel created in step 2.

In step 4, the TTP sends the delivery receipt to S. It is the certificate signed by the TTP's private signature key. The TTP is trusted to take this step jointly with the previous one, so as to be fair to both sender and receiver. If the certificate received inside the delivery receipt matches S's stored certificate, then S authenticates R.

The TTP sees the symmetric key k, but not the plaintext message m. This is a desirable goal that reduces the trust in the TTP: it cannot disclose the e-mails even if it later becomes compromised. (Note that a misbehaving TTP could eavesdrop on the initial message from S to R, taking the ciphertext $\{\!|m|\!\}_k$, which he could decrypt once he knows k.) Also, the use of encryption should prevent third parties from learning m. Most importantly, however, the protocol "should allow a sender, S, to send an e-mail message to a receiver, R, so that R reads the message if and only if S receives the corresponding return receipt" [2, §2].

4 Modelling Abadi et al.'s Protocol

Although the protocol does not require a public-key infrastructure, the TTP must have one key pair for encryption and another for signature. Hence, the protocol model requires a treatment of both kinds of asymmetric keys. The protocol also uses passwords to strengthen the authentication of the receiver to the TTP. Hence, a treatment of symmetric keys (shared with the TTP) is also necessary. A complete formalisation of asymmetric keys was developed to verify SET [6], and can be reused here. A formalisation of symmetric keys exists too, and has been used for a number of protocols relying on symmetric cryptogra-

```
FakeSSL:
 "⟦ evsfssl ∈ certified_mail; X ∈ synth(analz(knows Spy evsfssl))⟧
  ⟹ Notes TTP {|Agent Spy, Agent TTP, X|} # evsfssl ∈ certified_mail"

CM1:
 "⟦evs1 ∈ certified_mail;
   Key K ∉ used evs1;
   K ∈ symKeys;
   Nonce q ∉ used evs1;
   hs = Hash {|Number cleartext, Nonce q, response S R q,
               Crypt K (Number m)|};
   S2TTP = Crypt (pubEK TTP) {|Agent S, Number AO, Key K, Agent R, hs|}⟧
 ⟹ Says S R {|Agent S, Agent TTP, Crypt K (Number m), Number AO,
                Number cleartext, Nonce q, S2TTP|} # evs1
      ∈ certified_mail"

CM2:
 "⟦evs2 ∈ certified_mail;
   Gets R {|Agent S, Agent TTP, em', Number AO, Number cleartext',
            Nonce q', S2TTP'|} ∈ set evs2;
   TTP ≠ R;
   hr = Hash {|Number cleartext', Nonce q', response S R q', em'|} ⟧
 ⟹
  Notes TTP {|Agent R, Agent TTP, S2TTP', Key(RPwd R), hr|} # evs2
    ∈ certified_mail"

CM3:
 "⟦evs3 ∈ certified_mail;
   Notes TTP {|Agent R', Agent TTP, S2TTP'', Key(RPwd R'), hr'|}
     ∈ set evs3;
   S2TTP'' = Crypt (pubEK TTP)
                    {|Agent S, Number AO, Key k', Agent R', hs'|};
   TTP ≠ R';  hs' = hr';  k' ∈ symKeys⟧
 ⟹
  Notes R' {|Agent TTP, Agent R', Key k', hr'|} #
  Gets S (Crypt (priSK TTP) S2TTP'') #
  Says TTP S (Crypt (priSK TTP) S2TTP'') # evs3 ∈ certified_mail"
```

Fig. 5. Modelling Abadi et al.'s protocol (fragment)

phy [14]. We have added these extensions to our standard formalisation of public keys, where it can be used for verifying other protocols.

For authentication, R must be able to respond to a query q from S. The original protocol expects S and R to agree off-line on a series of challenge-response pairs. We chose the following implementation of responses:

$$\mathsf{response}\ S\ R\ q == \mathsf{Hash}\{\!|\mathsf{Agent}\ S, \mathsf{Key}(\mathsf{shrK}\ R), \mathsf{Nonce}\ q|\!\}$$

This explicit definition allows the spy to generate the response if R is compromised.

Fig. 5 shows the largest fragment of our protocol model, omitting only three classical rules (*Nil*, *Recp* and *Fake*, Fig. 2). Message transmission over a secure channel, which is authentic, confidential and delivery-guaranteed, is formalised by a Notes event of the form (2) discussed above (§2.2). Rule *FakeSSL* represents the possibility of the spy's opening a secure channel to TTP and sending a fake message. Rule *CM1* represents the first protocol message; note that *cleartext* is a part of the message that is given to R immediately. In rule *CM2*, a Notes event represents R's message to TTP; here $\mathsf{Key}(\mathsf{RPwd}\ R)$ is R's password. Steps 3 and 4 are assumed to take place at the same time, so they are formalised by the single rule *CM3*. The TTP checks R's password to authenticate the sender of message 2, but regardless he must reply along the same secure channel. The replies to both S and R are delivery-guaranteed, so the rule introduces an appropriate Notes event for the receiver, and a double Says-Gets event for the TTP's transmission to the sender. That Says event is unnecessary according to our simple formalisation of guaranteed delivery (§2.2), but is retained to preserve a feature of our model: a Gets event is always preceded by a matching Says event.

5 Verifying Abadi et al.'s Protocol

The novel features of this protocol required some new techniques in the verification. However, space limitations force us directly to the final results on certified e-mail delivery, also omitting the classical ones on confidentiality and authentication.

The guarantee for the sender is expressed as two theorems: one for the case when the recipient is the spy and one for honest recipients. The sender does not have to know which case applies.

Theorem 1 (Sender's guarantee: spy).

```
"⟦Says S R {|Agent S, Agent TTP, Crypt K (Number m), Number AO,
             Number cleartext, Nonce q, S2TTP|} ∈ set evs;
  S2TTP = Crypt (pubEK TTP) {|Agent S, Number AO, Key K, Agent R, hs|};
  Key K ∈ analz(knows Spy evs);
  evs ∈ certified_mail;
  S≠Spy⟧
⟹ R ∈ bad & Gets S (Crypt (priSK TTP) S2TTP) ∈ set evs"
```

This theorem's premises are that the sender has issued message 1 (with the given value of $S2TTP$) and that the session key is available to the spy. The conclusion is that R is compromised, but even in this case, the sender gets the return receipt.

Theorem 2 (Sender's guarantee: honest recipient).

```
"⟦Says S R {|Agent S, Agent TTP, Crypt K (Number m), Number AO,
             Number cleartext, Nonce q, S2TTP|} ∈ set evs;
  S2TTP = Crypt (pubEK TTP) {|Agent S, Number AO, Key K, Agent R, hs|};
  Notes R {|Agent TTP, Agent R, Key K, hs|} ∈ set evs;
  S≠Spy;  evs ∈ certified_mail⟧
 ⟹ Gets S (Crypt (priSK TTP) S2TTP) ∈ set evs"
```

In this version, the sender has issued message 1 and R has legitimately received the session key. The conclusion is that S gets the return receipt.

Theorem 3 (Recipient's guarantee).

```
"⟦Crypt (priSK TTP) S2TTP ∈ used evs;
  S2TTP = Crypt (pubEK TTP)
            {|Agent S, Number AO, Key K, Agent R,
              Hash {|Number cleartext, Nonce q, r, em|}|};
  hr = Hash {|Number cleartext, Nonce q, r, em|};
  R≠Spy;  evs ∈ certified_mail⟧
 ⟹ Notes R {|Agent TTP, Agent R, Key K, hr|} ∈ set evs"
```

The recipient's guarantee states that if the return receipt exists at all (as formalised by the function used) then R has received the session key.

The development of our proofs has highlighted that an anomalous execution of the protocol is possible. The receiver can initiate a session from step 2 by quoting an arbitrary sender, and by building two identical hashes. The session will terminate successfully and the sender will get evidence that an e-mail he has never sent has been delivered. This is due to the fact that the protocol uses no technique to authenticate the sender to TTP. The anomaly can be solved by inserting the sender's password into the certificate $S2TTP$ created at step 1, so that the receiver cannot forge it.

Another flaw is that S has no defence against R's claim that the message was sent years ago and is no longer relevant. This attack works in both directions: R's claim might be truthful and not believed. Even if S includes a date in the message, he cannot prove that the date is accurate. The obvious solution is for TTP to include a timestamp in the return receipt.

6 Conclusions

We have developed the concept of a second-level security protocol, namely one that uses a first-level protocol as a primitive. We have shown how correctness assertions for second-level protocols can be expressed. The existing primitives of the Inductive Approach already let us formalise such concepts as sending a

confidential message, an authenticated message, or a message with guaranteed delivery. As a concrete example, we have specified a certified e-mail protocol and formalised its correctness assertions. We have verified the main guarantees of this protocol.

Acknowledgements

Research was funded by the EPSRC grant GR/R01156/01 *Verifying Electronic Commerce Protocols*. Discussions with Martin Abadi, Bruno Blanchet, and Salvatore Riccobene were useful.

References

1. M. Abadi and B. Blanchet. Computer-Assisted Verification of a Protocol for Certified Email. In R. Cousot, editor, *Static Analysis, 10th International Symposium (SAS'03)*, Lecture Notes in Comp. Sci. Springer-Verlag, June 2003. To appear.
2. M. Abadi, N. Glew, B. Horne, and B. Pinkas. Certified email with a light on-line trusted third party: Design and implementation. In *Proceedings of the 11th International Conference on Wold Wide Web (WWW-02)*. ACM Press and Addison Wesley, 2002.
3. N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proc. of the 17th IEEE Sym. on Sec. and Privacy*. IEEE Comp. Society Press, 1998.
4. T. Aura. Distributed access-rights management with delegation certificates. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues with Distributed Mobile Objects*, volume 1603 of *Lecture Notes in Comp. Sci.*, pages 211–235. Springer-Verlag, 1999.
5. G. Bella. Inductive verification of smart card protocols. *J. of Comp. Sec.*, 11(1):87–132, 2003.
6. G. Bella, F. Massacci, and L. C. Paulson. Verifying the SET registration protocols. *IEEE J. of Selected Areas in Communications*, 21(1):77–87, 2003.
7. G. Bella and L. C. Paulson. Mechanical proofs about a non-repudiation protocol. In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2001*, volume 2152 of *Lecture Notes in Comp. Sci.*, pages 91–104. Springer, 2001.
8. B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. of the 14th IEEE Comp. Sec. Found. Workshop*. IEEE Comp. Society Press, 1998.
9. M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *Proceedings of the Royal Society of London*, 426:233–271, 1989.
10. J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Technical report, University of York, Department of Computer Science, November 1997. Available on the web at `http://www-users.cs.york.ac.uk/~jac/`. A complete specification of the Clark-Jacob library in CAPSL is available at `http://www.cs.sri.com/~millen/capsl/`.

11. E. Cohen. TAPS: A first-order verifier for cryptographic protocols. In *Proc. of the 13th IEEE Comp. Sec. Found. Workshop*, pages 144–158. IEEE Comp. Society Press, 2000.
12. F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand Spaces: Why is a Security Protocol Correct? In *Proc. of the 17th IEEE Sym. on Sec. and Privacy*. IEEE Comp. Society Press, 1998.
13. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems: second international workshop, TACAS '96*, volume 1055 of *Lecture Notes in Comp. Sci.*, pages 147–166. Springer, 1996.
14. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. of Comp. Sec.*, 6:85–128, 1998.
15. L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Trans. on Inform. and Sys. Sec.*, 2(3):332–351, 1999.
16. P. Y. A. Ryan and S. A. Schneider. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison Wesley Publ. Co., Reading, Massachussetts, 2000.
17. V. Shmatikov and J. C. Mitchell. Analysis of a fair exchange protocol. In *Network and Distributed System Security Symposium (NDSS-00)*, 2000.
18. J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *Symposium on Security and Privacy*. IEEE Computer Society, 1996.