Accountability Protocols: Formalized and Verified

Giampaolo Bella, Università di Catania
Lawrence C. Paulson, University of Cambridge

Classical security protocols aim to achieve authentication and confidentiality under the assumption that the peers behave honestly. Some recent protocols are required to achieve their goals even if the peer misbehaves. *Accountability* is a protocol design strategy that may help. It delivers to the peers sufficient evidence of each other's participation in the protocol. Accountability underlies the non-repudiation protocol of Zhou and Gollmann and the certified e-mail protocol of Abadi et al. This paper provides a comparative, formal analysis of the two protocols, and confirms that they reach their goals under realistic conditions. The treatment, which is conducted with mechanized support from the proof assistant Isabelle, requires various extensions to the existing analysis method. A byproduct is an account of the concept of *higher-level protocol*.

Categories and Subject Descriptors: F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Mechanical Verification*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol Verification*

General Terms: Security, Protocol, Verification

Additional Key Words and Phrases: Non-repudiation, Certified e-email, Proof tools, Inductive method, Isabelle

1. INTRODUCTION

Classical security protocols establish secure communications over insecure networks. Typically they ensure that no attacker can obtain sensitive information or impersonate another person. The protocol protects Alice and Bob, who trust one another, from hostile parties. This scenario is inappropriate when Alice does not even know Bob, let alone trust him. Purchasing goods over the Internet requires trusting the merchant with your credit card details, even if a protocol such as SSL protects against outsiders.

Preliminary registration is an attempt to strengthen trust. People who wish to participate must first enrol with an authority. Protocols that employ registration include SET [Mastercard & VISA 1997] and Visa 3-D Secure [VISA 2002]. Registration gives Alice some confidence in Bob—since he can present signed credentials—but it does not change the security framework. Alice still must trust Bob.

This work was funded by the EPSRC grant GR/R01156/R01 *Verifying Electronic Commerce Protocols*. Authors' addresses: Giampaolo Bella, Dipartimento di Matematica e Informatica, Università di Catania, Viale A. Doria 6, I-95125 Catania (ITALY), e-mail: giamp@dmi.unict.it; Lawrence C Paulson, Computer Laboratory, University of Cambridge, 15 JJ Thomson Avenue, Cambridge CB3 0FD (UK), e-mail: lcp@cl.cam.ac.uk

Accountability reduces the need for trust. The non-repudiation protocol of Zhou and Gollmann [Zhou and Gollmann 1996] sends a message while ensuring that neither the sender nor the receiver can deny taking part. Each of them receives sufficient evidence to prove the other's participation. The certified e-mail protocol by Abadi et al. [Abadi et al. 2002] similarly ensures that an e-mail is delivered if and only if its sender gets the return receipt. Both protocols are intended to achieve their goals even if the other party misbehaves: Alice need not trust Bob, and vice versa.

Our contribution is at least twofold. We describe general techniques for modelling and verifying accountability protocols inductively. We present a comparative treatment of the non-repudiation protocol [Zhou and Gollmann 1996] with the certified e-mail protocol [Abadi et al. 2002]. These results combine and supersede our previous works [Bella and Paulson 2001; Bella et al. 2003] as follows. Both non-repudiation and certified e-mail are recognised as forms of accountability, and the techniques for modelling and verifying them are unified and generalised accordingly. The realistic threat model where agents do not trust each other, which we developed to analyse the certified e-email protocol, is adopted to re-examine the non-repudiation protocol from scratch. In consequence, the formal guarantees about the latter now withstand peer agents who can send fake messages: an agent who obeys the protocol is protected from a possibly cheating peer.

We use our Inductive Method [Bella 2000; Paulson 1998] for protocol verification. Because the formalization is based on a general-purpose theorem prover [Nipkow et al. 2002], it is easily adapted to new security environments. Correctness of accountability protocols involves two concepts.

—*Validity of evidence*: an agent is given evidence sufficient to convince a third party of his peer's participation in the protocol.

—*Fairness*: both agents obtain the promised items, or neither do [Asokan et al. 1998a].

Proving the new properties required the development of novel strategies for proof and especially for specification. Allowing the peer to be the adversary could make proofs excessively complicated. To keep proofs simple, we must express the guarantees with care. We must also formalize various forms of *secure channels*: channels that satisfy properties such as authentication, confidentiality or guaranteed delivery. Many accountability protocols rely upon secure channels, which might be implemented by running another security protocol: we thus arrive at the concept of *higher-level protocols*. Accountability is often described in terms of evidence that can be presented to a judge. We model the evidence only, not the judge. Judges are human beings, and the assessment of a body of evidence requires reason and experience.

We have found both protocols to be correct: they are fair and they deliver valid evidence. More precisely, the non-repudiation protocol is fair in the sense that the initiator gets non-repudiation of receipt if and only if the responder gets non-repudiation of origin. Both pieces of evidence are proved valid. Along with his evidence, the responder also gets the message that the initiator intended to send him. The certified e-mail protocol achieves slightly weaker goals. It is fair in the sense that the initiator gets non-repudiation of receipt (a "return receipt") if and

only if the responder gets the e-mail. That receipt is proved valid. However, the responder gets no evidence for non-repudiation of origin. The e-mail itself does not suffice for this purpose.

This paper continues by presenting accountability protocols (§2) and how our method of analysis faced their challenges (§3). Then, the paper describes the inductive models (§4) and the formal guarantees (§5) for the new protocols. Finally, it concludes (§6).

2. ACCOUNTABILITY PROTOCOLS

An *accountability protocol* gives agents lasting evidence, typically digitally signed, about actions performed by his peer. Many authentication mechanisms fail to meet this requirement: the reply to an encrypted nonce challenge proves an agent's presence to the recipient but to nobody else. The protocol should meet its objectives to an honest agent even if the peer misbehaves.

This section presents the non-repudiation protocol [Zhou and Gollmann 1996] and the certified e-mail protocol [Abadi et al. 2002]. Both protocols have two peers and a trusted third party (TTP). The typical setting sees the sender's intention to transmit a message m to B. She encrypts it using a symmetric key k; the ciphertext $c = m_k$ is her *commitment* to the session with the receiver.

The private signature key of an agent X is indicated as sK_X^-, while the signature of a message y by key sK_X^- is $\{\!|y|\!\}_{sK_X^-}$. The public encryption key of an agent X is indicated as eK_X, while the encryption of a message y by key eK_X is $\{\!|y|\!\}_{eK_X}$. Our BAN-like notation [Burrows et al. 1989] makes no distinction between the operations of symmetric encryption, asymmetric encryption and signature, because the type of key suffices to disambiguate.

2.1 The non-repudiation protocol

The non-repudiation protocol (Fig. 1) uses a lightweight TTP whose effort is independent of the size of the transmitted message. A unique label, L, identifies the session between A and B. It concerns two types of evidence. Non-repudiation of origin (NRO) proves the participation of A, while non-repudiation of receipt (NRR) proves the participation of B. Flags such as f_{nro} express the non-repudiation meaning of a certificate.

In step 1, A picks a symmetric key k and a label L, and encrypts m with k to form c. Then, A signs f_{nro}, B, L, c to yield NRO, which she sends to B. In response (step 2), B verifies A's signature, signs f_{nrr}, A, L, c and sends the resulting NRR to A. Then (step 3), A lodges k with the TTP by sending sub_k, which is f_{sub}, B, L, k signed with her private signature key.

If the TTP can verify A's signature, it signs f_{con}, A, B, L, k producing con_k, which it makes available in its public directory. This step binds the key k to the session between A and B labelled L. Finally (steps 4 and 5), A and B download con_k from the TTP using the File Transfer Protocol (FTP); the protocol assumes this download will eventually succeed.

The protocol aims at providing both parties with evidence to prove the other's participation. The evidence for A consists of NRR and con_k, while that for B consists of NRO and con_k. Making con_k part of the evidence ensures fairness,

Abbreviations

$$
\begin{aligned}
c &= m_k \\
NRO &= \{\!|f_{nro}, B, L, c|\!\}_{sK_A^-} \\
NRR &= \{\!|f_{nrr}, A, L, c|\!\}_{sK_B^-} \\
sub\_k &= \{\!|f_{sub}, B, L, k|\!\}_{sK_A^-} \\
con\_k &= \{\!|f_{con}, A, B, L, k|\!\}_{sK_{\mathrm{TTP}}^-}
\end{aligned}
$$

Steps

1. $A \longrightarrow B \quad : f_{nro}, B, L, c, NRO$
2. $B \longrightarrow A \quad : f_{nrr}, A, L, NRR$
3. $A \longrightarrow \mathrm{TTP} : f_{sub}, B, L, k, sub\_k$
4. $B \overset{\mathsf{FTP}}{\longleftarrow} \mathrm{TTP} : f_{con}, A, B, L, k, con\_k$
5. $A \overset{\mathsf{FTP}}{\longleftarrow} \mathrm{TTP} : f_{con}, A, B, L, k, con\_k$

Fig. 1. The non-repudiation protocol

since TTP releases this item to both parties simultaneously.

Let us informally analyse how to resolve disputes. If A holds con_k and NRR, then she has completed a run with B, who has accepted the commitment c and should be able to download the decryption key from the TTP. Similarly, if B holds con_k and NRO, then A cannot deny having sent c as a commitment bound to label L. Of course, such arguments are unconvincing: we need formal verification.

2.2 The certified e-mail protocol

Abadi et al. [Abadi et al. 2002] have designed a realistic protocol for certified e-mail delivery. No public-key infrastructure is necessary: the TTP has signature and encryption keys, but other agents merely share a password with the TTP. Agent R's password is indicated pwd_R. In common with the previous protocol, the TTP is lightweight and its effort is independent of the e-mail size; moreover, this TTP is stateless. A challenge-response mechanism authenticates the receiver to the sender, who must agree beforehand on some acknowledgement function linking a challenge q to its response r.

As in the non-repudiation protocol, the sender forms a commitment by encrypting his e-mail with a symmetric key, k. He attaches k, encrypted with the TTP's public encryption key. The recipient forwards the message to the TTP in order to obtain k. (In the non-repudiation protocol, the sender lodges the key directly with the TTP.) The TTP releases the key and simultaneously releases a certificate documenting the transaction to the sender.

We present the full version of the protocol (Fig. 2), where the receiver authenticates to both the sender and the TTP. In step 1, the sender S sends the receiver R the encrypted e-mail c, a challenge q, and a certificate for the TTP, called $S2TTP$. The certificate is encrypted under the TTP's public encryption key and contains the symmetric key k that protects c along with a hash linking c to the required response, r. Recall that R and S must have already agreed a query-response mechanism.

In step 2, R computes the response r to the query q and includes the received ciphertext c to build the hash h_R, which he sends along with the received certificate

Abbreviations

$$\begin{aligned} c &= m_k \\ h_S &= hash(q, r, c) \\ h_R &= hash(q, r, c) \\ S2TTP &= \{\!|S, k, R, h_S|\!\}_{eK_{\mathrm{TTP}}} \\ RR &= S2TTP_{sK^{-}_{\mathrm{TTP}}} \end{aligned}$$

Steps

1. $S \longrightarrow R$: TTP, $c, q, S2TTP$
2. $R \xrightarrow{\mathsf{SSL}}$ TTP : $S2TTP, pwd_R, h_R$
3. TTP $\xrightarrow{\mathsf{SSL}} R$: k, h_R
4. TTP $\longrightarrow S$: RR

Fig. 2. The certified e-mail protocol

and his password (pwd_R) to the TTP on a secure channel. The authors state that security here means confidentiality and authentication, and that "in practice, such a channel might be an SSL connection" [Abadi et al. 2002]. They also require guaranteed delivery, which can be implemented by sending a message repeatedly until it is acknowledged [Deng et al. 1996].

In step 3, the TTP decrypts and verifies the received certificate. Then, the TTP authenticates R by the password and—to check that S and R agree on the authentication mechanism—verifies that h_S found inside the ticket matches h_R. 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 a signed return receipt RR to S. Note that RR is essentially non-repudiation of receipt (NRR). The TTP must take this step jointly with the previous one, so as to be fair to both sender and receiver. If the certificate received inside the return receipt matches S's stored certificate, then S authenticates R.

In both protocols, the TTP sees the symmetric key k, but not the plaintext message m. This reduces the trust in the TTP, which cannot disclose the e-mails even if compromised. However, 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.

The protocol's use of encryption should prevent spies from learning m. Most importantly, 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" [Abadi et al. 2002, §2]. This objective is similar to that of Zhou and Gollmann, but weaker. The responder does not receive non-repudiation of origin (NRO), namely evidence that the initiator intended to send him the message. Nenadiċ et al. [Nenadic et al. 2004] have recently published an e-mail protocol that provides non-repudiation of both origin and receipt.

3. ANALYSING ACCOUNTABILITY PROTOCOLS

The verification of security protocols is well understood. Many techniques exist [Blanchet 1998; Cohen 2000; Fábrega et al. 1998; Paulson 1998; Ryan and

Schneider 2000] to verify a wide variety of protocols, often automatically. However, accountability protocols raise new challenges. We build upon our existing work on analysing security protocols using the Inductive Method [Bella 2003; Paulson 1998] and the proof assistant Isabelle/HOL.

3.1 Isabelle

Isabelle is a general-purpose interactive theorem prover. It is *generic*, which means that it can reason in a variety of formal systems. The best-developed and most popular version is Isabelle/HOL [Nipkow et al. 2002]; it supports *higher-order logic*, a typed formalism that allows quantification over functions, predicates and sets. Hardware and software systems can readily be modelled in higher-order logic, and correctness properties expressed.

Isabelle provides much automation. Its *simplifier* (`simp`) combines rewriting with arithmetic decision procedures, and it also has automatic provers (`auto`, `blast`, `force`, etc.). However, most proofs are done interactively. In a typical proof, the user directs Isabelle to perform a certain induction and then to simplify the resulting subgoals. Any surviving subgoals might be given to an automatic prover or reduced to other subgoals by the use of some lemma. Failure to find a proof may help in locating a bug in the system being modelled, but it could simply mean that the user is not skilled enough.

The series of commands used to prove a theorem can be seen as a proof sketch. Confidence that the proof is sound comes from observing that line of reasoning that we are forced to adopt, and the lemmas we are forced to prove. Interactive theorem proving is difficult, and this very difficulty strengthens our confidence that the resulting theorems are true. Conversely, a fully automatic proof may lead to worries that the model of our system is too abstract. When a theorem has been proved, Isabelle can deliver a formal proof object; as automation improves, people will increasingly want to examine these proof objects (perhaps using an independent tool) in order to assure themselves that the proof is valid.

Security protocols can be modelled using general-purpose tools such as Isabelle. Once the models have been shown to be useful, researchers will naturally use them as the basis for specialized protocol verification tools, which may achieve high performance. Isabelle remains useful for modelling novel protocols that are not covered by the specialized tools.

3.2 The Inductive Method

Our general approach is to formalise the system inductively. The resulting operational semantics has much in common with CSP formalisations [Ryan and Schneider 2000], 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. These are lists (built in reverse order) containing three kinds of *events*.

—`Says A B X` means that `A` attempts to send message `X` to `B`.

—*Gets B X* means that *B* receives message *X* from the network.

—*Notes A X* means that *A* stores message *X* in its local state.

Messages are a recursive datatype *msg* that includes the following constructors:

—*Agent A* denotes the name of the agent *A*.

—*Number N* denotes a guessable number, where *N* is a non-negative integer, of type *nat*.

—*Nonce N* denotes a non-guessable number, such as a random byte string, of type *nat*.

—*Key K* denotes a key, which is regarded as non-guessable, of type *key*.

—*Hash X* denotes the cryptographic hash of the message *X*.

—*Crypt K X* denotes encryption of message *X* with key *K*.

—{|*X1,..., Xn*|} denotes the concatenation of the messages {|*X1, ..., Xn*|}.

Protocol definitions 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.

—*knows A evs* denotes the set of messages that *A* sends or receives in the trace *evs*. If *A* is the spy, then the set comprises all messages anyone sends or receives in the trace.

—*parts H* denotes the set of all message components that appear in the set of messages *H*, including the plaintexts of all encryptions.

—*analz H* is a subset of *parts H*, the components that are effectively derivable from *H* using decryption by derivable keys.

—*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).

In the following simple protocol, an initiator sends her identity and a fresh nonce to a responder, who replies by signing the nonce.

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

Its inductive model (Fig. 3) consists of four introduction rules defining the constant *flp*, the set of traces permissible with this protocol. Rule *Nil* admits the empty trace. Rule *Fake* allows the spy to generate a message *X* using material gleaned from past traffic and send it to anyone. Rule *flp1* represents the first protocol step: *A* chooses a fresh nonce and sends it to *B*. Rule *flp2* represents the second protocol step: if *B* receives a suitable message, he signs the nonce and sends it to

the agent named in the first component of the message he received. Rule *Recp* says that if a message is sent, it might be received.

```
Nil:  "[] ∈ flp"

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

FLP1: "⟦evs1 ∈ flp; Nonce Na ∉ used evs1⟧
       ⟹ Says A B {|Agent A, Nonce Na|} # evs1 ∈ flp"

FLP2: "⟦evs2 ∈ flp; Gets B {|Agent A, Nonce Na|} ∈ set evs2⟧
       ⟹ Says B A (Crypt (priSK B) (Nonce Na)) # evs2 ∈ flp"

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

Fig. 3. The inductive model of a simple protocol

3.3 Formalizing and verifying the novel goals

Non-repudiation and certified e-mail delivery are similar forms of accountability, and differ from more basic concepts such as confidentiality and authentication. The crucial difference is that protocol goals are proved without trusting the peer to be well-behaved.

There are two pen-and-paper analyses of Zhou and Gollman's protocol. One, by the authors themselves [Zhou and Gollmann 1998], uses the SVO authentication logic to only reason about validity of evidence. The other, by Schneider [Schneider 1998], uses rank functions and CSP to reason both about validity of evidence and about fairness. An automated analysis by Gürgens and Rudolph finds a replay attack under the assumption that the TTP does not maintain an audit trail: the initiator can re-use the supposedly unique session label, using evidence from a past run to "prove" the responder's participation in a recent session [Gürgens and Rudolph 2002].

We find the Gürgens-Rudolph scenario unrealistic.[1] An audit trail is fundamental to security. The protocol authors require some state to be maintained:

> We require the TTP to check that the keys provided to it do not overwrite existing entries in the public directory." [Zhou and Gollmann 1996, §5.2].

They also comment on the duration of record-keeping:

> In practice, we will not want TTP to store message keys forever. We could set a deadline T to limit the time con_k and k can be accessed by the public." [Zhou and Gollmann 1996, §5.3].

[1] Gollman, in a private communication, has rejected it emphatically.

We would expect the TTP to keep an off-line log of its transactions for a much longer period than T, so that disputes can be investigated.

The certified e-mail protocol is fairly recent, and we know of only one formal analysis. Abadi and Blanchet [2003] have used the protocol verifier ProVerif and conclude, as we do, that the protocol meets its goals.

However, none of the aforementioned analyses are directly useful to us, as the technical problems are specific to the verification method. We have developed simple formalizations of non-repudiation and certified e-mail delivery, along with simple proof strategies relying on induction.

The goal of non-repudiation is that at the end of a protocol session the initiator has `NRR` and the responder has `NRO`. Given a trace `evs` of the protocol model, the goals can be expressed as `NRR ∈ analz(knows A evs)` and `NRO ∈ analz(knows B evs)`. (Recall that the `analz` operator extracts components from the given set of messages by decomposing concatenated messages and decrypting ciphertexts using available keys.) If the protocol also is fair, then either both goals or none of them are achieved. The goal of a *fair* non-repudiation protocol can be expressed abstractly using a logical equivalence:

$$\texttt{NRO} \in \texttt{analz(knows B evs)} \iff \texttt{NRR} \in \texttt{analz(knows A evs)}$$

Here `evs` ranges over all protocol traces, and we do not assume the remote agent to be honest. Certified e-mail delivery can be expressed similarly: if the receiver can derive the e-mail from his view of the network traffic on `evs`, then the sender can derive the corresponding return receipt from his view of the traffic, and vice versa.

Given a specific protocol, these logical equivalences must be refined. Their symmetry is misleading: it does not capture the protocol's objectives. The operator `analz` represents an unlimited amount of work, and is only appropriate when specifying a hostile party. The honest agent expects to be given the required items directly. The two directions must be expressed as separate implications (here, for certified e-mail):

—If the (untrusted) receiver can derive the e-mail (using `analz`), then the sender has been given the return receipt.

—If the (untrusted) sender can derive the return receipt (using `analz`), then the receiver has been given the e-mail.

The phrase "has been given" must be formalized by reference to the specific protocol message that delivers the required item. Thus, we can only formalize correctness with respect to a specific protocol.

An additional problem is that we currently lack techniques for reasoning about the knowledge of agents other than the spy. Specifically, we do not know how to prove any interesting consequences of the assumption `X ∈ analz(knows A evs)` unless `A` is the spy. Sometimes, we can correct the situation by proving something stronger that no longer involves `analz`. Here is an improved guarantee for the recipient of a certified e-mail delivery:

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

This guarantee is stronger than the original, while eliminating the need to reason about the sender's knowledge.

The same technique can be applied to the messages `NRO` and `NRR` used in non-repudiation. Unfortunately, it cannot be used to express the sender's guarantee of certified e-mail: obviously the e-mail `m` will have been created. Instead, we can divide that guarantee into two separate assertions. One assumes that the recipient is the spy; it is proved by reasoning about `analz(knows Spy evs)`, which we know how to do. The other assumes that the recipient is honest, and states that if `R` is given `m` then `S` is given the receipt.

Both of our protocols use a key to protect their transmitted message. Later, the responder (receiver) is given this key, so that she can decrypt the message. By referring to this key, we can further refine our guarantees. A statement such as "`R` has the message" can be refined to "`R` has the key to the message." We can also formalise a further guarantee, namely that the key never reaches the spy. This valuable secrecy guarantee is formalised and proved using standard methods.

3.4 Higher-level protocols

Classical security protocols rely on basic transport protocols to deliver cryptographic messages. The non-repudiation protocol refers to a specific transport protocol, FTP. The certified e-mail protocol depends on a secure channel, which might be established using SSL [Abadi et al. 2002]. A security protocol that relies on other security protocols suggests the concept of *higher-level* security protocol. Other examples include the fair-exchange protocol by Asokan et al. [Asokan et al. 1998b], which relies on an authentication protocol, and Visa 3-D Secure [VISA 2002], which relies on SSL.

This concept is independent of accountability and deserves future investigation. Here is a formal definition.

Definition 1 Higher-Level Protocols.

—*A* 0^{th}-level protocol *is a transport protocol.*

—*An* i^{th}-level protocol, $i \geq 1$ *is a protocol that uses cryptography and* j^{th}*-level protocols for* $0 \leq j < i$.

An i^{th}-level protocol, $i \geq 1$, is a security protocol. Our certified e-mail protocol is at level two; our non-repudiation protocol is at level one, since FTP is only a transport protocol. A protocol's level is a property of its design and not of its capabilities: any security protocol can be reduced to level one by expanding the definitions of the protocols it uses. This obviously happens when the protocol is implemented, when security analysts must look out for unexpected interactions among the various protocols.

3.5 Formalizing the underlying protocols

Formalizing a second-level protocol requires a formalization of the underlying first-level protocols as black boxes. Expanding their definitions is impractical, and for the certified e-mail protocol, the use of SSL is merely a suggestion. The main properties required of the first-level protocol are authentication and confidentiality, and sometimes guaranteed delivery. Below we describe how to model these proper-

ties abstractly in our inductive framework: in other words, how to model channels secured by first-level protocols.

3.5.1 *Authentication.* The sender identity `A` of a `Says A B X` event designates the true sender of the message. We normally do not allow the recipient to inspect the sender identity, formalizing message reception using the `Gets` event. In earlier work [Paulson 1998], we formalized message reception using the event `Says A' B X`, taking care to ensure that the value of `A'` (the true sender) was never used.

If `B` can authenticate the sender, then we can simply relax these restrictions in the corresponding part of the inductive definition. For example, `Says A B X` signifies 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 `Says A B X`.

3.5.2 *Confidentiality.* What if the channel must be confidential? We could extend our definitional framework with an event `ConfSays A B X` for sending a message confidentially. This change would require modifications throughout our modelling framework, and it is unnecessary: the framework is already expressive enough to model secure channels.

The `Notes` event formalises an agent's changing his internal state. It has the form `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 specific event

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

Notice that the identities of the peers are stored with the actual message by convention. Let us consider a simple second-level protocol that modifies our first-level protocol (§3.2) to send its first message using SSL, which we model as a confidential and authenticated channel:

1. $A \xrightarrow{\mathsf{SSL}} B : A, Na$
2. $B \longrightarrow A : \{\!|Na|\!\}_{sK_B^-}$

The first message is formalized by rule `SLP1` (Fig. 4). A new rule, `Recp1st`, formalizes the reception of the confidential message: reception is not guaranteed even on a confidential channel. This new rule takes event 1 as a precondition and introduces the event

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

signifying that `B` receives `X` confidentially. The rule `SLP2`, which formalizes `B`'s actions upon reception of `X`, takes event 2 as a precondition. Here `A` denotes the true sender of the message, which `B` knows thanks to the secure channel. Rule `Fake1st` is motivated later (§3.6). It is like the usual `Fake` rule except that it introduces the event `Notes B` $\{\!|$`Spy,B,X`$|\!\}$, formalizing `B`'s reception of the fake message `X` on an authenticated channel. In other words, as `Fake` models the spy's ability to send fake messages over transport protocols, `Fake1st` models the same over first-level protocols.

3.5.3 *Guaranteed Delivery.* Other goals of first-level protocols can be formalized similarly. Guaranteed delivery can be formalized by introducing the event

```
Nil:     "[] ∈ slp"

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

Fake1st: "⟦evsf1 ∈ slp; X ∈ synth(analz(knows Spy evsf1))⟧
          ⟹ Notes B {|Agent Spy, Agent B, X|} # evsf1 ∈ slp

SLP1:    "⟦evs1 ∈ slp; Nonce Na ∉ used evs1⟧
          ⟹ Notes A {|Agent A, Agent B, Nonce Na|} # evs1 ∈ slp"

SLP2:    "⟦evs2 ∈ slp;
           Notes B {|Agent A, Agent B, Nonce Na|} ∈ set evs2⟧
          ⟹ Says B A (Crypt (priSK B) (Nonce Na)) # evs2 ∈ slp"

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

Recp1st: "⟦evsr1 ∈ slp; Notes A {|Agent A, Agent B, X|} ∈ set evsr1⟧
          ⟹ Notes B {|Agent A, Agent B, X|} # evsr1 ∈ slp" "
```

Fig. 4. The inductive model of a second-level protocol

for receiving a message at the same time as (or instead of) the event for sending it. To combine this goal with confidentiality, as it would be needed for example to confidentially distribute a session key, we can use an inductive rule that gives both recipients `Notes` that have the form seen above and contain that key. Clearly, the formalization of guaranteed delivery never uses reception rules such as `Recp` or `Recp1st` (Fig. 4).

`Notes` events are affected by a detail of our model, the set `bad` of *compromised agents*. 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 to assert that the spy is a member and the TTP is not.

We have presented simple methods of formalizing secure channels abstractly—that is, we do not assume a specific underlying protocol such as SSL or Kerberos. To avoid conflicts with this use of `Notes`, other `Notes` messages should not begin with two agent names.

3.6 Defining and formalizing a threat model

The usual threat model formalised in the Inductive Method is a Dolev-Yao spy. He monitors the network traffic by means of function `Knows`, can analyse that traffic by means of function `analz`, and can synthesise new messages from the analysed components by means of function `synth`. The spy can send such messages to anyone, as formalised by rule `Fake` (Fig. 4). Our present work requires extensions to this threat model:

(1) *Guarantees must not assume the peer to be honest.* This is the main difference between accountability protocols and traditional ones. This principle affects

the formalization of guarantees but not the formalization of the protocol.

(2) *The spy can use channels created by first-level protocols, for example, SSL channels.* This principle requires the additional rule `Fake1st` in the protocol definition, as discussed above.

(3) *The spy cannot break the first-level protocols.* This principle has guided our design of the techniques for modelling secure channels. We assume that a secure channel is always secure.

4. FORMALIZING THE PROTOCOLS

The techniques developed above allow us to inductively define the non-repudiation protocol (§4.1) and the certified e-mail protocol (§4.2) .

We build on the Isabelle theory `Public`[2] for cryptographic protocols based on asymmetric encryption. It provides separate key pairs for encryption and signature. Each agent also has a long-term symmetric key, which we use to model the passwords of the e-mail protocol. The encryption primitive, `Crypt`, expresses symmetric and asymmetric encryption and also digital signature.

4.1 Formalizing the non-repudiation protocol

The protocol model is the set of traces `zg`, whose inductive definition is in Fig. 5. It is built according to the template given above in Fig. 3. Rules `Nil`, `Fake` and `Reception` are standard. Rules `ZG1`, `ZG2`, `ZG3` and `ZG4` respectively model the legitimate protocol steps. In particular, to initiate the protocol with `B`, agent `A` chooses a fresh label in rule `ZG1`. "Labels have to be unique to create the link between commitment and key." [Zhou and Gollmann 1996, §5.2], so we decide to model them as random numbers, namely as nonces. Therefore, our labels are independent of the messages—we do not study more detailed computations of labels. Because `A` sends the message `m` in an encrypted form, she must choose a cryptographic key. Rule `ZG1` leaves her free to choose any key, even an old one; we merely assume that she cannot pick asymmetric keys. Rule `ZG4` gives `con_K` to the spy: since the TTP places this value on an FTP site, we model the possibility of the spy's downloading it.

We highlight the important certificates by defining them in the premises, using equations; we use the names so defined in the conclusions. When a certificate is defined in the premises of a rule, then the rule only applies for a certificate of the specified form: informally, the agent verifies it. For example, `B` must check that `NRO` in rule `ZG2` is signed by `A` in order to confirm that she is the sender of the message just received. Likewise, `A` must check that `NRR` in rule `ZG3` is signed by `B`.

Rule `ZG4` models `TTP`'s preparation of the key confirmation `con_k`. The `TTP` verifies the signature on `sub_k` to confirm the identities of the other agents. All the components needed to verify that signature are available. We decide to model the installation of `con_k` in `TTP`'s public directory by a `Notes` event. This step terminates the protocol with the availability of `con_k` from `TTP`: we do not need to model the peers' retrieval of `con_k` via FTP-get. However, because the spy only knows compromised agents' notes and `TTP` is not compromised (§3.5.3), the spy is formally

[2]Found in `src/HOL/Auth` of the Isabelle distribution

```
Nil:  "[] ∈ zg"

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

Reception: "⟦evsr∈zg; Says A B X ∈ set evsr⟧ ⟹ Gets B X # evsr ∈ zg"

ZG1: "⟦evs1 ∈ zg;  Nonce L ∉ used evs1; C = Crypt K (Number m);
       K ∈ symKeys;
       NRO = Crypt (priK A) {|Number f_nro, Agent B, Nonce L, C|}⟧
   ⟹ Says A B {|Number f_nro, Agent B, Nonce L, C, NRO|} # evs1 ∈ zg"

ZG2: "⟦evs2 ∈ zg;
       Gets B {|Number f_nro, Agent B, Nonce L, C, NRO|} ∈ set evs2;
       NRO = Crypt (priK A) {|Number f_nro, Agent B, Nonce L, C|};
       NRR = Crypt (priK B) {|Number f_nrr, Agent A, Nonce L, C|}⟧
   ⟹ Says B A {|Number f_nrr, Agent A, Nonce L, NRR|} # evs2  ∈  zg"

ZG3: "⟦evs3 ∈ zg; C = Crypt K M; K ∈ symKeys;
       Says A B {|Number f_nro, Agent B, Nonce L, C, NRO|} ∈ set evs3;
       Gets A {|Number f_nrr, Agent A, Nonce L, NRR|} ∈ set evs3;
       NRR = Crypt (priK B) {|Number f_nrr, Agent A, Nonce L, C|};
       sub_K = Crypt (priK A) {|Number f_sub, Agent B, Nonce L, Key K|}⟧
   ⟹ Says A TTP {|Number f_sub, Agent B, Nonce L, Key K, sub_K|}
          # evs3 ∈ zg"

ZG4: "⟦evs4 ∈ zg; K ∈ symKeys;
       Gets TTP {|Number f_sub, Agent B, Nonce L, Key K, sub_K|}
         ∈ set evs4;
       sub_K = Crypt (priK A) {|Number f_sub, Agent B, Nonce L, Key K|};
       con_K = Crypt (priK TTP) {|Number f_con, Agent A, Agent B,
                                  Nonce L, Key K|}⟧
   ⟹ Says TTP Spy con_K
       # Notes TTP
              {|Number f_con, Agent A, Agent B, Nonce L, Key K, con_K|}
       # evs4 ∈ zg"
```

Fig. 5. Formalizing the non-repudiation protocol

not allowed to download *con_K*. To circumvent this limitation, rule *ZG4* introduces a *Says* event whereby that message is explicitly revealed to the spy.

4.2 Formalizing the certified e-mail protocol

The protocol model is the set of traces *certified_mail*, part of whose inductive definition appears in Fig. 6. (We omit rules *Nil*, *Fake* and *Reception*.) It is built according to the template given above in Fig. 4. For authentication, *R* must be able to respond to a query *q* from *S*. The two agents should have agreed off-line on a series of challenge-response pairs. We choose the following implementation of responses, which allows the spy to generate the response if *R* is compromised—

though not if *S* is compromised.

```
"response"    :: "agent => agent => nat => msg"
"response S R q == Hash {|Agent S, Key (shrK R), Nonce q|}"
```

According to the general treatment given above (§3.5), message transmission over a secure channel, which is authenticated, confidential and delivery guaranteed, is formalized by a *Notes* event of the form (2). Rule *Fake1st* lets the spy open a secure channel to the TTP and send a fake message. Rule *CM1* represents the first protocol message; *cleartext* stands for the part of the message that is given to *R* immediately. In rule *CM2*, a *Notes* event represents *R*'s message to the TTP; here *Key (RPwd R)* is *R*'s password. Because messages 2 and 3 travel over guaranteed-delivery channels, the protocol model does not require a rule of the form of *Recp1st* (§3.5.3). Hence, the subjects of the *Notes* events in rules *CM2* and *CM3* respectively are the intended recipients of the messages (*TTP* and *R* respectively).

Steps 3 and 4 must take place at the same time, so they are formalized 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. The *Says* event may seem unnecessary, but it preserves a feature of our model: every *Gets* event has a matching *Says* event.

5. VERIFYING THE PROTOCOLS

In previous work, we have described how to prove authentication and secrecy properties for a variety of protocols [Bella 2000; 2003; Paulson 1998]. Here we focus on novel techniques, primarily on how to specify and verify guarantees where the peer may be the spy. We list all the important guarantees, but only outline the proofs of the novel ones. To convey an impression of the verification process, we briefly discuss the machine proofs.

5.1 Verifying the non-repudiation protocol

For verifying this protocol, we require an additional definition: the set of *broken* agents. It is necessary because if an agent's signing key has been compromised, signatures made using that key are worthless. The set *broken* therefore includes all compromised agents other than the spy.

```
broken == bad - {Spy}
```

If an agent is broken, then the spy has his keys and can impersonate him freely, so many protocol guarantees assume the peer to be unbroken. This assumption still allows the peer to be the spy himself, and therefore to misbehave.

All of the guarantees are *regularity* properties, which concern all well-formed protocol runs. We do not have to prove difficult properties involving secrecy. In message 3, agent *A* actually broadcasts the key *K*.

5.1.1 *Proving validity of evidence.* A lemma states that if *con_K* exists at all (as formalized by the function *used*), then *TTP* has stored it on the FTP site, where it is available to *A* and *B*. (Our model does not include the actual FTP-get operations.)

```
Fake1st: "⟦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 BothAuth, Key K, Agent R, hs|}⟧
      ⟹ Says S R {|Agent S, Agent TTP, Crypt K (Number m), Number BothAuth,
                    Number cleartext, Nonce q, S2TTP|} # evs1
          ∈ certified_mail"

CM2: "⟦evs2 ∈ certified_mail;
       Gets R {|Agent S, Agent TTP, em, Number BothAuth, 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 BothAuth, 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. 6. Formalizing the certified e-mail protocol

Either agent, possessing *con_K*, can use this guarantee to show that the peer has access to *con_K*, and therefore to the key *K*. Since *con_K* is equally available to both parties, this lemma also expresses an aspect of fairness.

lemma *con_K_validity:*

```
     "⟦con_K ∈ used evs;
       con_K = Crypt (priK TTP)
                  {|Number f_con, Agent A, Agent B, Nonce L, Key K|};
       evs ∈ zg⟧
   ⟹ Notes TTP {|Number f_con, Agent A, Agent B, Nonce L, Key K, con_K|}
          ∈ set evs"
```

The proof is a simple induction: since *con_K* is signed by *TTP*, who is uncompromised, rule *ZG4* must have been executed.

The Isabelle proof script is six lines long. The first three lines, which are routine, set up the induction.

apply *clarify*

```
apply (erule rev_mp)
apply (erule zg.induct)
```

The fourth line is also routine. It applies *ZG2_msg_in_parts_spies*, a typical *forwarding lemma* [Paulson 1998, §4.2], and then it simplifies all subgoals arising from the induction. In this case, the forwarding lemma causes Isabelle to note that message component *C*, which gets incorporated into *NRR*, has been transmitted in clear and is therefore already known to the spy.

```
apply (frule_tac [5] ZG2_msg_in_parts_spies, simp_all)
```

Only two subgoals survive the simplification. The first, which arises from the *Fake* rule, is proved by *blast* with the help of a lemma that concerns the relationship between faked messages and the *parts* primitive. The remaining case concerns rule *ZG2*; it is also proved by *blast* with a different lemma concerning *parts*.

```
apply (blast dest!: Fake_parts_sing_imp_Un)
apply (blast dest: parts_cut)
```

Finding such proofs requires ingenuity, but often one proof script can serve as the starting point for the next one.

Our first proper theorem states that if *con_K* exists, then *A* has sent a well-formed instance of message 3. Its conclusion holds even if *A* is the spy, which is allowed by *A* ∉ *broken*.

```
theorem B_sub_K_validity:
     "⟦con_K ∈ used evs;
       con_K = Crypt (priK TTP) {|Number f_con, Agent A, Agent B,
                                  Nonce L, Key K|};
       sub_K = Crypt (priK A) {|Number f_sub, Agent B, Nonce L, Key K|};
       A ∉ broken; evs ∈ zg⟧
 ⟹ Says A TTP {|Number f_sub, Agent B, Nonce L, Key K, sub_K|} ∈ set evs"
```

The proof script consists of a single line.

```
by (blast dest: con_K_validity Notes_TTP_imp_Says_A)
```

Here *Notes_TTP_imp_Says_A* is a lemma, proved by induction, stating that if *con_K* is on the FTP site then *A* has sent message 3.

We have also proved that *NRO* is valid. This proof requires some lemmas to be proved beforehand; let us examine the development in detail. First, we prove that if *NRO* appears to come from *A*, then it really comes from *A*. The easy case is when *A* is uncompromised (*A* ∉ *bad*): the spy could not have made the digital signature, so it arose from a legitimate protocol step.

```
lemma NRO_validity_good:
     "⟦NRO = Crypt (priK A) {|Number f_nro, Agent B, Nonce L, C|};
       NRO ∈ parts (knows Spy evs);
       A ∉ bad;  evs ∈ zg⟧
     ⟹ Says A B {|Number f_nro, Agent B, Nonce L, C, NRO|} ∈ set evs"
```

The Isabelle proof script is simple—as we expect for such an elementary claim—and consists of a standard induction setup followed by *auto*.

```
apply clarify
apply (erule rev_mp)
apply (erule zg.induct)
apply (frule_tac [5] ZG2_msg_in_parts_spies, auto)
```

The following lemma states that if anybody has sent any messge resembling *NRO* and involving *A*'s signature, then the sender was either *A* or the spy. The proof is another easy induction, since only the spy would use another agent's key.

```
lemma NRO_sender:
     "⟦Says A' B {|n, b, 1, C, Crypt (priK A) X|} ∈ set evs;  evs ∈ zg⟧
      ⟹ A' ∈ {A,Spy}"
```

Thus, we arrive at our second theorem: if *A'* has sent an instance of *NRO* signed by *A*, then *A* has also sent the message. If *A'* ≠ *A* then (by the previous lemma) *A' = Spy*, hence *A* ≠ *Spy*. The result follows by *NRO_validity_good*.

```
theorem NRO_validity:
     "⟦Gets B {|Number f_nro, Agent B, Nonce L, C, NRO|} ∈ set evs;
       NRO = Crypt (priK A) {|Number f_nro, Agent B, Nonce L, C|};
       A ∉ broken;  evs ∈ zg⟧
     ⟹ Says A B {|Number f_nro, Agent B, Nonce L, C, NRO|} ∈ set evs"
```

The two theorems presented so far, namely *B_sub_K_validity* and *NRO_validity*, confirm the protocol goals for *B*. If he exhibits *con_K*, then by theorem *sub_K_validity* he can assert that *A* submitted the key *K* bound to the label *L*. If he exhibits *NRO*, then by theorem *NRO_validity* he can assert that *A* submitted the commitment *C* bound to the label *L*. The label binds the commitment to the key, hence the theorems together confirm *A*'s intention to send the plaintext message contained in *C*.

An analogous theorem, with a similar proof, guarantees that *NRR* is valid. Any instance of *NRR* that appears to come from *B* actually did. As usual, the assumption *B* ∉ *broken* allows *B* to be the spy.

```
theorem NRR_validity:
     "⟦Gets A {|Number f_nrr, Agent A, Nonce L, NRR|} ∈ set evs;
       NRR = Crypt (priK B) {|Number f_nrr, Agent A, Nonce L, C|};
       B ∉ broken; evs ∈ zg⟧
    ⟹ Says B A {|Number f_nrr, Agent A, Nonce L, NRR|} ∈ set evs"
```

This theorem, together with *sub_K_validity*, confirms the protocol goals for *A*. If *A* exhibits *NRR* and *con_K*, then she can assert that *B* holds *C* and *K*, and therefore has access to *m*.

5.1.2 *Proving fairness.* The fairness guarantees protect an agent who follows the protocol from one who does not. The person receiving the guarantee must be uncompromised, but no assumption is made about the peer. Since *con_k_validity* already states that *con_K* is equally available to both parties, we only have to prove fairness for *NRO* and *NRR*.

This theorem expresses fairness for *B*: if *NRR* exists at all, then *B* (who must be uncompromised) holds *NRO*. The proof is yet another straightforward induction.

```
theorem B_fairness_NRR:
     "⟦NRR ∈ used evs;
```

```
    NRR = Crypt (priK B) {|Number f_nrr, Agent A, Nonce L, c|};
    NRO = Crypt (priK A) {|Number f_nro, Agent B, Nonce L, c|};
    B ∉ bad; evs ∈ zg⟧
  ⟹ Gets B {|Number f_nro, Agent B, Nonce L, c, NRO|} ∈ set evs"
```

Fairness for `A` has a slightly different form: if `con_K` and `NRO` exist, then `A` holds `NRR`. We see how `con_K` gives fairness to `A`, who otherwise would be at a disadvantage because the first message gives evidence to `B`.

theorem *A_fairness_NRO:*

```
  "⟦con_k ∈ used evs;
    NRO ∈ parts (knows Spy evs);
    con_k = Crypt (priK TTP)
                  {|Number f_con, Agent A, Agent B, Nonce L, Key k|};
    NRO = Crypt (priK A) {|Number f_nro, Agent B, Nonce L, Crypt k m|};
    NRR = Crypt (priK B) {|Number f_nrr, Agent A, Nonce L, Crypt k m|};
    A ∉ bad;  evs ∈ zg⟧
  ⟹ Gets A {|Number f_nrr, Agent A, Nonce L, NRR|} ∈ set evs"
```

The Isabelle proof is much more complicated than that of the corresponding property for `B`. Four cases (`Fake`, `ZG1`, `ZG2` and `ZG4`) require separate attention. We need a lemma that `A` only sends message 3 after she has received `NRR`. She recognizes the correct `NRR` by the label `L`, which she is required to choose uniquely to identify the transaction. This uniqueness is used in the proof of fairness for `A`. If `A` attempts to cheat by re-using transaction identifiers, as suggested by Gürgens and Rudolph [Gürgens and Rudolph 2002], then she runs the risk of accepting the wrong transaction.

5.2 Verifying the certified e-mail protocol

The novel features of this protocol required low-level modifications to the verification techniques, but little that was fundamentally new. We focus on the final results on certified e-mail delivery, omitting proofs of the classical properties of confidentiality and authentication. We present three theorems, which together confirm the main goal of the protocol, that the sender `S` gets the return receipt if and only if the receiver `R` gets the e-mail. As before, we include some crucial lemmas and details about the proof scripts.

5.2.1 *Proving validity of evidence.* This protocol offers no protection against non-repudiation of origin, so there is no evidence given to the receiver. The main theorem confirming the validity of evidence says that if the return receipt exists, then `R` has obtained the cryptographic key necessary to retrieve the e-mail.

theorem *RR_validity:*

```
  "⟦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 cryptogram mentioned in the first line is the return receipt. The inductive proof is lengthy (eleven commands), with separate consideration of four cases of the induction. Nothing inherently difficult is involved; the complicated form of the assertion causes Isabelle's automatic provers to require more guidance than usual. In the terminology of the Inductive Method, this proof relies on forwarding and regularity lemmas [Paulson 1998], which are elementary facts about the message structure.

Before proceeding to the fairness guarantees, we need to introduce a lemma. It concerns *S2TTP*, the certificate sent by *S* to *TTP* in the first protocol message. The lemma says that anything matching the form of *S2TTP* can only arise from a valid instance of the first protocol message, provided the spy does not know the key *K* within it.

```
lemma S2TTP_sender:
 "⟦Crypt (pubEK TTP) {|Agent S, Number AO, Key K, Agent R, hs|} ∈ used evs;
   Key K ∉ analz (knows Spy evs);
   evs ∈ certified_mail⟧
 ⟹ ∃m ctxt q.
      hs = Hash{|Number ctxt, Nonce q, responseSR q, Crypt K (Number m)|} &
      Says S R
         {|Agent S, Agent TTP, Crypt K (Number m), Number AO,
          Number ctxt, Nonce q,
          Crypt (pubEK TTP)
            {|Agent S, Number AO, Key K, Agent R, hs|}|} ∈ set evs"
```

The proof is straightforward—the spy needs to know *K* before he can use it to make a fake version of *S2TTP*—but once again the proof script includes quite a bit of guidance for Isabelle's provers.

Using this lemma requires satisfying the premise *Key K ∉ analz (knows Spy evs)*. That can be done in several ways. We could use a separate proof that *K* is confidential, or perform a case analysis on whether *K* is confidential or not. Finally, we could leave the confidentiality as an outstanding assumption, to be proved later.

The lemma *Key_unique*, not shown, is related to *S2TTP_sender*. It again assumes that the spy does not know *K*; it concludes that if two instances of the first protocol message agree in component *K*, then they agree in all components. It is a typical unicity theorem [Paulson 1998, §4.5], with a straightforward proof. It holds because honest senders do not reuse keys.

5.2.2 *Proving fairness.* The theorem *RR_validity* above expresses fairness for *R*: he will get the required key if *S* gets the return receipt. The fairness guarantee for *S* is expressed as two theorems: one for the case when the receiver is the spy and one for an honest receiver. The sender does not need to know which case applies.

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

```
theorem S_fairness_bad_R:
     "⟦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);
  S ≠ Spy; evs ∈ certified_mail⟧
⟹ R ∈ bad & Gets S RR ∈ set evs"
```

The proof script is a simple induction except for the treatment of the third protocol message, when *TTP* replies to *S* and *R*. Here, the reasoning is rather subtle. Any assertion of the form "if the spy knows *K* then ..." is a confidentiality property. (In particular, if ... is simply *False* then it is equivalent to saying that the spy does not know *K*.) Therefore, the proof requires the sort of reasoning needed to prove confidentiality [Paulson 1998, §4.5]. We get a case analysis on whether the spy knows the key or not, and in the latter case we appeal to lemmas *S2TTP_sender* and *Key_unique*. Even with this complicated argument, Isabelle's provers do much of the work, and the treatment of the third message consists of only five commands.

In the "good" theorem, the sender has issued message 1 and the receiver has legitimately received the session key. The conclusion is that the sender gets the return receipt.

```
theorem S_guarantee:
 "⟦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"
```

Here, the Isabelle proof script is surprisingly short. The argument for the crucial third message consists of a single prover call:

```
apply(blast dest: Notes_SSL_imp_used S2TTP_sender Key_unique S_fairness_bad_R)
```

However, this generates a rather intricate proof involving the lemmas discussed above.

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 because the protocol does not 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, which would devalue the return receipt. 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.

The formalization of a distrusted peer differs from the previous protocol, where we had to assume $A \notin broken$ rather than just $A \neq Spy$. The non-repudiation protocol requires the stronger assumption because it is based on digital signatures, which are worthless if the peer's private keys have been disclosed. The certified e-mail protocol is based upon weaker mechanisms: passwords and previously-agreed

responses. We must bear that in mind when interpreting the theorems proved in our model. Guarantees based on strong cryptography are firmer than those based on weak passwords.

It can be interesting to see what happens if the protocol is deliberately weakened. For example, suppose that the receiver of the certified e-mail protocol forgets to send message 2 over the SSL protocol and sends it over a conventional transport protocol. Modelling this variant is straightforward: replace the event

```
Notes TTP {|Agent R, Agent TTP, S2TTP, Key(RPwd R), hr|}
```

in rule *CM2* by the event

```
Says R TTP {|S2TTP, Key(RPwd R), hr|}
```

and fix the corresponding premise in rule *CM3* accordingly. An attempt to re-execute our proof script soon reveals that the receiver is sending his secret password in clear and disclosing it to the spy. Specifically, the regularity lemma stating that the spy only knows the passwords of compromised agents fails in the new model.

This kind of experiment is interesting, but unnecessary. The Inductive Method works by establishing facts through formal proof. The chain of reasoning is open to inspection. This is fundamentally different from model checkers and other automatic analysis tools. Bug-finding tools often use unrealistic finite models and they yield no justification of their result if they fail to find a bug. Hence comes the impulse to inject bugs, or to strengthen the protocol specification, so that the tool can report something. Some of these findings are even published as if they were realistic attacks.

6. CONCLUSIONS

Our findings confirm that both the non-repudiation protocol and the certified e-mail protocol broadly achieve their goals. The non-repudiation protocol delivers evidence to its participants, binding each other's participation. It is fair: each party receives evidence if and only if the other party does. The certified e-mail protocol is fair in the sense that the initiator gets non-repudiation evidence—the return receipt—if and only if the responder gets the e-mail. All evidence in both protocols appears to be valid: sufficient to hold an agent accountable for participation.

Comparing the two protocols, the e-mail one demands less of the trusted third party and it uses much weaker cryptographic mechanisms, with no public-key infrastructure. It offers correspondingly weaker guarantees: the responder gets no non-repudiation evidence, and even the theorems we can prove must be interpreted with an awareness of the weak cryptography.

Our methods scale up to analysing accountability protocols. We have examined two protocols that have similar goals but operate in very different security environments. Numerous, though straightforward, changes were necessary to model the novel architectures.

REFERENCES

Abadi, M. and Blanchet, B. 2003. Computer-assisted verification of a protocol for certified email. In *Static Analysis, 10th International Symposium (SAS'03)*, R. Cousot, Ed. Lecture Notes in Comp. Sci., vol. 2694. Springer Verlag, 316–335.

Abadi, M., Glew, N., Horne, B., and Pinkas, B. 2002. 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.

Asokan, N., Shoup, V., and Waidner, M. 1998a. Asynchronous protocols for optimistic fair exchange. In *Proc. of the 17th IEEE Sym. on Sec. and Privacy*. IEEE Comp. Society Press, 86–99.

Asokan, N., Shoup, V., and Waidner, M. 1998b. Asynchronous protocols for optimistic fair exchange. In *Proc. of the 17th IEEE Sym. on Sec. and Privacy*. IEEE Comp. Society Press.

Bella, G. 2000. Inductive Verification of Cryptographic Protocols. Ph.D. thesis, Research Report 493, Computer Laboratory, University of Cambridge. Accepted for publication as LNCS Monograph by Springer.

Bella, G. 2003. Inductive verification of smart card protocols. *J. of Comp. Sec. 11,* 1, 87–132.

Bella, G., Longo, C., and Paulson, L. C. 2003. Verifying second-level security protocols. In *Theorem Proving in Higher Order Logics: TPHOLs 2003*, D. Basin and B. Wolff, Eds. LNCS 2758. Springer, 352–366.

Bella, G. and Paulson, L. C. 2001. Mechanical proofs about a non-repudiation protocol. In *Theorem Proving in Higher Order Logics: TPHOLs 2001*, R. J. Boulton and P. B. Jackson, Eds. Lecture Notes in Comp. Sci., vol. 2152. Springer, 91–104.

Blanchet, B. 1998. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. of the 14th IEEE Comp. Sec. Found. Workshop*. IEEE Comp. Society Press.

Burrows, M., Abadi, M., and Needham, R. M. 1989. A logic of authentication. *Proceedings of the Royal Society of London 426*, 233–271.

Cohen, E. 2000. TAPS: A first-order verifier for cryptographic protocols. In *Proc. of the 13th IEEE Comp. Sec. Found. Workshop*. IEEE Comp. Society Press, 144–158.

Deng, R. H., Gong, L., Lazar, A. A., and Wang, W. 1996. Practical protocols for certified electronic mail. *Journal of Network and System Management 4,* 3, 279–297.

Fábrega, F. J. T., Herzog, J. C., and Guttman, J. D. 1998. Strand Spaces: Why is a Security Protocol Correct? In *Proc. of the 17th IEEE Sym. on Sec. and Privacy*. IEEE Comp. Society Press.

Gürgens, S. and Rudolph, C. 2002. Security analysis of (un-) fair non-repudiation protocols. In *Formal Aspects of Security*, A. Abdallah, P. Ryan, and S. Schneider, Eds. Technical Report CSD-TR-02-13.

Mastercard & VISA 1997. *SET Secure Electronic Transaction Specification: Business Description*. Mastercard & VISA. On the Internet at `http://www.setco.org/set_specifications.html`.

Nenadic, A., Zhang, N., and Barton, S. 2004. Fair certified e-mail delivery. In *Proc. of the 18th ACM Symposium on Applied Computing (ACM SAC'04)*. ACM Press and Addison Wesley, 391–396.

Nipkow, T., Paulson, L. C., and Wenzel, M. 2002. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer. LNCS Tutorial 2283.

Paulson, L. C. 1998. The inductive approach to verifying cryptographic protocols. *J. of Comp. Sec. 6*, 85–128.

Ryan, P. Y. A. and Schneider, S. A. 2000. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison Wesley Publ. Co., Reading, Massachussetts.

Schneider, S. 1998. Formal analysis of a non-repudiation protocol. In *11th Computer Security Foundations Workshop*. IEEE Computer Society Press, 54–65.

VISA 2002. *3-D Secure Introduction*. VISA. On the Internet at `http://international.visa.com/fb/paytech/secure/pdfs/3DS_70001-01_Introduction_v1.0.2.pdf`.

Zhou, G. and Gollmann, D. 1998. Towards verification of non-repudiation protocols. In *International Refinement Workshop and Formal Methods Pacific*, J. Grundy, M. Schwenke, and T. Vickers, Eds. Springer-Verlag, 370–380.

Zhou, J. and Gollmann, D. 1996. A fair non-repudiation protocol. In *Symposium on Security and Privacy*. IEEE Computer Society.