Verification of Security Protocols
Chapter 7: Non-repudiation, Fairness and an Analysis of a Certified E-mail Protocol

Christian Haack

April 7, 2008
Plan for Today

- We will discuss non-repudiation and fairness.
- We will discuss a verification of a certified e-mail protocol with ProVerif (based on an article by Abadi and Blanchet [AB05]).
Certified E-mail Protocols

- Alice wants to send a message to Bob, but she does not want him to read it without her obtaining a receipt.
- A postal worker handles this in real life.
- How can this be handled over the network?
Threat Model

Legitimate participants may cheat.
- E.g., for certified e-mail:
  - receivers may deny having received e-mails (e.g., bills)
  - agents may forge e-mail receipts to falsely claim they have sent messages

This threat model is different from what we considered so far.
- **So far:** We have analyzed runs between honest agents in the presence of external attackers (external threat model) and spies inside the system (internal threat model).
- **Today:** We analyze runs between agents that try to cheat each other.
Non-repudiation protocols protect agents from cheating each other.

They provide the agents with evidence that certain steps in the protocol have occurred.

"Evidence" consists of a set of messages that (in case of dispute) can be given to a judge to prove that the evidenced protocol step has happened.
A protocol guarantees **non-repudiation of origin (NRO)** if it provides evidence to receivers that the message has been sent by the claimed sender.

- NRO can be achieved by the sender digitally signing the message.

A protocol guarantees **non-repudiation of receipt (NRR)** if it provides evidence to senders that the intended receiver has received the message.

- One may think that NRR can be achieved by a signed receipt. But ... this assumes an honest receiver who sends the receipt. So this does not work.

**Example:** Certified e-mail protocols aim to achieve non-repudiation of receipt.
Fairness

- A protocol guarantees **fairness** if no participant can gain advantage over another one by holding the protocol part way through.

  **Example:**

  \[
  A \rightarrow B : \text{mail} \\
  B \rightarrow A : \text{receipt}
  \]

  Unfair, because $B$ may hold the protocol after receiving the *mail*, leaving $A$ with no *receipt*.

  **Example:**

  $A$ generates fresh key $k$

  \[
  A \rightarrow B : \{\text{mail}\}_k \\
  B \rightarrow A : \text{receipt} \\
  A \rightarrow B : k
  \]

  Unfair, because $A$ may hold the protocol after receiving the *receipt*, leaving $B$ unable to read the *mail*. 
Trusted Third Parties and Reliable Channels

- To achieve non-repudiation and fairness, protocols often make use of a trusted third party (TTP).
- In contrast to the other agents, the TTP is assumed to be honest.
- Non-repudiation protocols usually have to assume that the channels from the TTP to other agents deliver messages reliably in the sense that messages from the TTP eventually reach their destination. (In particular, adversaries cannot intercept messages on these channels.)
- There are protocols that work without a TTP and achieve fairness by sending messages bit by bit, in order to ensure that at every protocol stage each agent knows at most one bit more than the other agent. We will not look at such protocols in this class.
The Zhou–Gollmann Protocol (ZGP): Goals

Goals:

- A sends B a message $M$.
- ZGP provides non-repudiation of receipt to A.
- ZGP provides non-repudiation of origin to B.
- ZGP provides fairness in the following sense: At each protocol stage either both $A$ and $B$ have their evidences or neither of them has it.

A protocol that achieves all of these goals is called a fair non-repudiation protocol.
Channel 3 is assumed to guarantee reliable message delivery. In particular, it is assumed that B cannot block this channel to prevent delivery of message 3.
Channels 4 and 5 are assumed to guarantee reliable message delivery. They are meant to be "fetch-channels", where A and B retrieve the signed keys from a public read-only directory. (Think of ftp_get.) Messages 4 and 5 may happen in opposite order.
The Zhou–Gollmann Protocol

$nro, nrr, sub, con$: message tags
$sA, sB, sT$: signing keys of $A, B, TTP$
$k$: key generated by $A$ to construct a commitment $\{M\}_k$
$n$: nonce serving as a run identifier

1. $A \rightarrow B$ : $A, \{nro, B, n, \{M\}_k\}_sA$
2. $B \rightarrow A$ : $\{nrr, A, n, \{M\}_k\}_sB$
3. $A \rightarrow TTP$ : $\{sub, B, n, k\}_sA$
   $TTP$ computes $CON = \{con, A, B, n, k\}_sT$
4. $TTP \rightarrow B$ : $CON$
5. $TTP \rightarrow A$ : $CON$

- $A$’s evidence of receipt: $\{nrr, A, n, \{M\}_k\}_sB$ and $CON$
- $B$’s evidence of receipt: $\{nro, B, n, \{M\}_k\}_sA$ and $CON$
The Zhou–Gollmann Protocol: Properties

Roughly, ZGP has the following properties:

- At the end of a complete run:
  - $A$ has her evidence iff $B$ has received message $M$. \textit{(NRR)}
  - $B$ has his evidence iff $A$ has sent message $M$. \textit{(NRO)}

- At each protocol stage:
  - Either both $A$ and $B$ have their evidence or neither of them has. \textit{(Fairness)}

Zhou–Gollmann’s paper [ZG96] does not present a formal analysis that proves such properties. (But some formal analyses of ZGP exist, e.g., by Schneider using CSP as a modeling language [Sch98], and by Bella and Paulson using Isabelle as an interactive prover to produce machine-checked proofs [BP06].)
Certified E-mail with a Light On-line TTP

We will now consider a protocol by Abadi and Glew presented at the World Wide Web Conference in 2002 [AGHP02].

Characteristics.

- The protocol uses an on-line TTP.
  - On-line vs. off-line TTP.
    - an on-line TTP participates in every protocol run
    - an off-line TTP is only consulted in case of dispute
- The on-line TTP is "light" in the sense that it only learns a key, but never the entire content of an e-mail. The TTP’s computation overhead does not depend on the size of e-mail messages but only on the number of messages sent (similar to ZGP).
- The protocol does not require a public key infrastructure. Authentication of e-mail clients is password-based.
The message flow is similar to ZGP.

However, because this protocol provides NRR only (and not NRO), the message from $R$ back to $S$ is missing, and $R$ (instead of $S$) sends the key to the TTP.
Message Flow and Assumptions on Channels

1. encrypted message to R
2. encrypted key to TTP
3. key
4. receipt

Channels 3 and 4 are assumed to provide reliable message delivery.
Message Flow and Assumptions on Channels

Channel 2 and 3 are assumed to provide the following:
- server authentication of TTP
- confidentiality
Server authentication and confidentiality are, for instance, provided by SSL/TLS. So we can think of channels 2 and 3 as a TLS connection, where TTP acts as a TLS server.
The Protocol

\[ pk : \text{TTP’s public encryption key} \]
\[ sk : \text{TTP’s signing key} \]
\[ pwd : \text{password to authenticate } R \text{ to } \text{TTP} \]
\[ q/r : \text{query/response pair to authenticate } R \text{ to } S \]

\[ em \overset{\Delta}{=} \{ m \}_k \]
\[ h \overset{\Delta}{=} \text{hash}(q, r, em) \]
\[ S2TTP \overset{\Delta}{=} \{ S, k, R, h \}_pk \]
\[ \text{receipt} \overset{\Delta}{=} \{ S2TTP \}_{sk} \]

\( S \) generates key \( k \)

1. \( S \rightarrow R : \text{TTP, em, q, S2TTP} \)
2. \( R \rightarrow \text{TTP} : S2TTP, pwd, h \)
3. \( \text{TTP} \rightarrow R : k, h \)
4. \( \text{TTP} \rightarrow S : \text{receipt} \)
Protocol Analysis

- The certified e-mail protocol has been analyzed by Abadi and Blanchet using ProVerif [AB05].
- Their analysis makes heavy use of ProVerif, but some parts of the analysis are done manually.
- Fortunately, the harder parts are done with ProVerif.
Declarations

(* Hashing. *)
fun hash/1.

(* Symmetric crypto. *)
fun encrypt/2.
reduc decrypt(encrypt(x,y), y) = x.

(* Public key crypto. *)
fun pencrypt/2.
fun enc/1.
reduc pdecrypt(pencrypt(x, enc(y)), y) = x.

(* Signing. *)
fun sign/2.
fun dec/1.
reduc unsign(sign(x, y), dec(y)) = x.
Declarations (cont.)

(* Secure password lookup function. *)
private fun passwd/1.

(* Secure response lookup function, maps agent id and query to response. *)
private fun reply/2.

(* Public channel. *)
free net.

(* Agent names (simple model with fixed agents). *)
free S,R,TTP.

(* Private keys (simple model with fixed agents). *)
private free dpk, sk.

(* Sender’s query (simple model with fixed agents). *)
free q.
A Channel with Private Input Capabilities

- The channel from client \( R \) to server \( TTP \) provides server authentication. This means:
  - \( TTP \) is the only agent that can receive on this channel.
  - However, any agent can send on this channel.

- Here is how we model this in ProVerif:

  (* Authenticated channel to TTP. *)
  private free to_ttp.
  (* This alone would make to_ttp’s output capability private, too. *)

  ... | (!in(net,x); out(to_ttp,x)) | ...
  (* Now any message published on net can be relayed to to_ttp. This way, to_ttp’s output capability becomes public. *)
The Sender

let processS =
in(net, receiver);
new m; new k;
let em = encrypt(m,k) in
let r = reply(receiver,q) in
let h = hash( (q,r,em) ) in
let S2TTP = pencrypt( (S,k,receiver,h), pk ) in
out(receiver, (TTP, em, q, S2TTP));
! (* loop in case we get garbage *)
in(S, receipt);
let =S2TTP = unsign(receipt,dsk) in
  0
else
  out(S, receipt) (* put garbage back on channel *)
.

- Note that we use agent ids as channel names.
- In order to facilitate reliable message delivery from TTP, the process loops until it receives the receipt.
Modeling the TLS Connection

- The TLS connection is modeled as a pair of channels from $R$ to $TTP$, and from $TTP$ to $R$.
- To create these channels, $R$ generates a new name `socket`.
- The `socket` is sent to $TTP$ over the channel `to_ttp` (whose input capability is private).
- To send messages from $R$ to $TTP$, the pair $(TTP, socket)$ is used as a channel.
- To send messages from $TTP$ to $R$, the pair $(R, socket)$ is used as a channel.
- Because the `socket` is only known to $R$ and $TTP$, these channels provide confidentiality.
let processR =
  in(R, (=TTP, em, q, S2TTP));
  (* Begin establishing secure connection to TTP. *)
  new socket;
  let inchannel = (R, socket) in
  let outchannel = (TTP, socket) in
  out(to_ttp, inchannel);
  (* End establishing secure connection to TTP. *)
  let h = hash( (q, reply(R,q), em) ) in
  out(outchannel, (S2TTP, passwd(R), h));
  in(inchannel, (k, =h));
  let m = decrypt(em, k) in 0.
let processTTP =
  (* Begin establishing a secure connection with R. *)
  in(to_ttp, outchannel);
  let (receiver, socket) = outchannel in
  let inchannel = (TTP, socket) in
  (* End establishing a secure connection with R. *)
  in(inchannel, (S2TTP, pwd, h));
  if pwd = passwd(receiver) then
    let (sender, k, =receiver, =h) = pdecrypt(S2TTP, dpk)
    out(outchannel, (k, h));
    out(sender, sign(S2TTP, sk)).
The System

process
  let pk = enc(dpk) in out(net,pk);
  let dsk = dec(sk) in out(net,dsk);
  ( (!processS) | (!processR) | (!processTTP) |
    (!in(net,x); out(to_ttp,x)) )
For a start, we can now query for secrecy of message $m$:

query attacker : $m$.

ProVerif proves that $m$ remains secret.
In order to analyze if the protocol satisfies non-repudiation of receipt, we add a process that models a judge who checks receipts for their validity:

```
let processJ =
  in(J, (receipt,sender,k,receiver,q,r,em));
let S2TTP = unsign(receipt,dsk) in
let S2TTP' =
  pencrypt( ( sender,k,receiver,hash((q,r,em)) ), pk) in
if S2TTP = S2TTP' then
let m = decrypt(em,k) in
event SaysReceived(receiver,m).
```
Presenting Evidence

We add an output statement to the sender, presenting evidence to the judge:

```ocaml
let processS =
  ...
  let =S2TTP = unsign(receipt,dsk) in
  out(J, (receipt,S,k,receiver,q,r,em) )
else
  ...
  .
```

We also have to activate the judge:

```ocaml
process
  ...
  | (!processJ) | ...
```
Formalizing Non-repudiation

1. $S \rightarrow R : TTP, \text{em}, q, S2TTP$
2. $R \rightarrow TTP : S2TTP, \text{pwd}, h$
3. $TTP \rightarrow R : k, h$
   
   *$R$ issues event $\text{Received}(R, m)$*
4. $TTP \rightarrow S : \text{receipt}$
5. $S \rightarrow J : \text{receipt}, S, k, R, q, r, \text{em}$
   
   *$J$ issues event $\text{SaysReceived}(R, m)$*

Want to show (roughly):

$J$ issues $\text{SaysReceived}(R, m)$ if and only iff $R$ issues $\text{Received}(R, m)$.

More precisely:

1. Any continuation of a trace that executes $\text{Received}(R, m)$ eventually executes $\text{SaysReceived}(R, m)$.
2. Any continuation of a trace that executes $\text{SaysReceived}(R, m)$ eventually executes $\text{Received}(R, m)$. 
Limitations of ProVerif

- ProVerif cannot prove either of statement (1) or (2) directly. Why?
- ProVerif supports **backwards reasoning**, but no **forwards reasoning**:
  - ProVerif can prove this:
    - If event $E$ gets executed, then $E_1, \ldots, E_n$ must have got executed before.
  - ProVerif **cannot** prove this:
    - If event $E$ gets executed, then $E'$ will eventually get executed in the future.

- Furthermore: statements (1) and (2) only hold under the assumption that the channels for messages 3, 4 and 5 guarantee eventual message delivery. ProVerif cannot take such assumptions into account.
How Can we Still Employ ProVerif?

- We factor each of properties (1) and (2) into a backwards and a forwards property, by inserting auxiliary events.
- The backwards properties can be proven automatically with ProVerif (and without any assumption of eventual message delivery).
- The forwards property must be proven manually (and requires the assumptions of eventual message delivery).
- Fortunately, the proof of the backwards property is the hard part.

We detail this method for statement (1). Recall it:

Any continuation of a trace that executes Received(R,m) eventually executes SaysReceived(R,m).
Inserting Auxiliary Events

A.  
   S issues event SendingMsg(S, k, R, q, r, em)

1.  
   S → R : TTP, em, q, S2TTP

2.  
   R → TTP : S2TTP, pwd, h

B.  
   TTP issues event SendingReceipt(S2TTP)

3.  
   TTP → R : k, h

C.  
   R issues event Received(R, m)

4.  
   TTP → S : receipt

5.  
   S → J : receipt, S, k, R, q, r, em

D.  
   J issues event SaysReceived(R, m)

Backwards part for statement (1):

If C gets executed,
then A and B got executed before.

Forwards part for statement (1):

Any continuation of a trace that executes A and B eventually executes D.
Modeling a Dishonest Receiver

- For statement (1), we need to assume that receiver $R$ is dishonest.
- We therefore replace the receiver process with a process that publishes $R$'s secrets, and issues $\text{Received}(R, x)$ on any message $x$ that it gets its hands on:

\begin{verbatim}
(* Modeling a dishonest receiver. *)
let processR =
 (* publish all secrets *)
  out(net, to_ttp)
| out(net, passwd(R))
| out(net, reply(R,q))
 (* issue Received(R,x) for any message I can see *)
| in(net, x); event Received(R,x).
\end{verbatim}
Manual Proof of Forwards Part

The forwards part is quite easily proven manually, using the assumptions that messages 4 and 5 are delivered reliably, and assuming that output is asynchronous. Here is a proof sketch:

- Because A got executed, S is waiting for an input as provided by message 4.
- Because B got executed, TTP outputs message 4.
- Because channel 4 delivers messages reliably, S will eventually input message 4.
- Because the argument to events A and B match up, the equality test that S does in between receiving 4 and sending 5 succeeds.
- Because channel 5 delivers messages reliably, J will eventually input message 5.
- Because the arguments of events A and B match up, the equality tests and destructor applications between receiving 5 and issuing D succeed.
More ProVerif Queries

ProVerif allows queries of the following form:

query ev : M ==> ev : N1 & ... & ev : Nk;

Meaning:

If event $M$ gets executed,
then events $N_1, \ldots, N_k$ got executed before.
query

\[
\text{ev : Received}(R',m') \implies \\
\text{ev : SendingMsg}(S',k',R',q',r',\text{encrypt}(m',k')) \\
& \text{ev : SendingReceipt}(
   \text{pencrypt(}
   ( S',k',R', \text{hash( (q',r',\text{encrypt}(m',k')) ) })
   pk'
   )
).
\]

- The events in this query look slightly more complicated than the events $C$, $B$ and $A$ from the earlier slide.
- This is because I have expanded abbreviations like $em$ and $S2TTP$.
- Furthermore, I have replaced concrete names by variables (hence the primes).
Unfortunately, the proof of this query fails with the following attack:

\[ l \rightarrow R : m \]
\[ \text{event } \text{Received}(R, m) \]

We need to assume that the intruder does not know \( m \) initially!

To this end, we use a trick. We declare a private tag:

\[ \text{private fun sec/1.} \]

We modify the protocol in one and only one spot:

\[ \text{let } \text{processS} = \ldots \text{ let } \text{em} = \text{encrypt(sec(m),k) in } \ldots \]

We have tagged \( m \) with \( \text{sec} \).

Because the \( \text{sec} \)-tag is private, the intruder does not know \( \text{sec}(m) \) initially.

Because the \( \text{sec} \)-tag is never inspected at runtime, this modification does not change the protocol behaviour. It is therefore a legitimate modification.
Second Attempt

- We also have to \texttt{sec}-tag all occurrences of \texttt{m'} in the query:

```
query
  ev : Received(R',sec(m')) ==> 
    ev : SendingMsg(S',k',R',q',r',encrypt(sec(m'),k')) 
& ev : SendingReceipt(
    pencrypt(
      ( S',k',R',hash((q',r',encrypt(sec(m'),k')))
    pk'
  )
).
```

- Now, ProVerif successfully verifies the query. Hooray!
Recall the two halves of NRR:

1. Any continuation of a trace that executes $\text{Received}(R,m)$ eventually executes $\text{SaysReceived}(R,m)$.
2. Any continuation of a trace that executes $\text{SaysReceived}(R,m)$ eventually executes $\text{Received}(R,m)$.

- We have succeeded showing the first of these statements by factoring it in a (manually proved) forwards property and an (automatically proved) backwards property.
- The proof of the second statement uses the same general strategy.
- If interested, take a look at Abadi and Blanchet’s paper [AB05] and try to prove the second statement yourself.
Today we have seen a fair non-repudiation protocol and a certified e-mail protocol.

Although ProVerif cannot reason about non-repudiation directly, we have seen how, in the case of the certified e-mail protocol, it can be used to automate a large part of the reasoning.
Announcements

- There is a long break to the next class. The next class will be on May 19.
- Don’t miss the grand opening of EIPSI on Monday/Tuesday April 21,22! With talks by a number of world-famous security experts, including Whitfield Diffie and Bruce Schneier.
Today’s class was based on the following articles:

- “A Fair Non-repudiation Protocol” by Zhou and Gollmann [ZG96].
- “Certified E-mail with a Light On-line Trusted Third Party: Design and Implementation” by Abadi, Glew, Horne and Pinkas [AGHP02].
- “Computer-assisted Verification of a Protocol for Certified E-mail” by Abadi and Blanchet[AB05].


