Computer-Assisted Verification of a Protocol for Certified Email

Martín Abadi
University of California, Santa Cruz

Bruno Blanchet
Département d'Informatique, École Normale Supérieure
and Max-Planck-Institut für Informatik

June 2003

Case study: A certified email protocol

A certified email protocol:

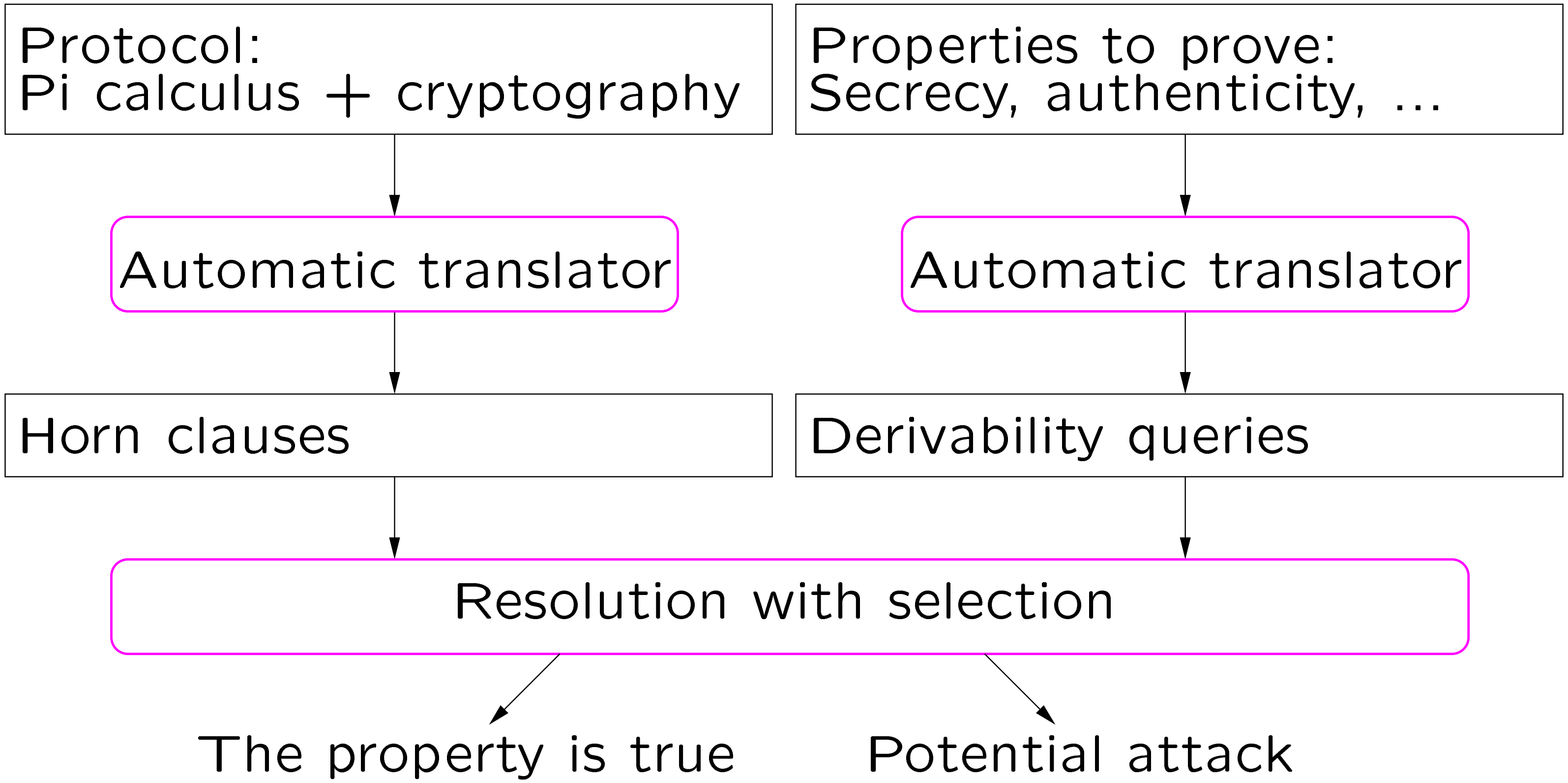
- sends an electronic message from a sender S to a receiver R,
- such that S has a receipt if and only if R has the message.

We prove the main security properties of the certified email protocol of Abadi, Glew, Horne, and Pinkas [WWW'02], relying on an automatic verifier.

(See also new work by Bella, Longo, Paulson with Isabelle.)

The protocol is fairly non-trivial and representative.

Our verification technique



Features of our technique

- Automatic
- Fast
- For unbounded number of sessions and message space
- For a wide range of cryptographic primitives

Correspondence assertions [see Woo and Lam, 1993]

We instrument the protocol with events $\mathbf{begin}(M)$ and $\mathbf{end}(M)$.

We can then prove properties concerning the occurrences of these events.

In previous work, we showed how to prove correspondence assertions of the form:

> If $\mathbf{end}(M)$ has been executed,
> then $\mathbf{begin}(M)$ must have been executed.

Correspondence assertions (new developments)

We instrument the protocol with events $\mathbf{begin}(M)$ and $\mathbf{end}(M)$.

We can then prove properties concerning the occurrences of these events.

To verify this protocol, we prove more general correspondence assertions of the form:

> If some event $\mathbf{end}(M)$ has been executed,
> then the events $\mathbf{begin}(M_1)$, ..., $\mathbf{begin}(M_n)$ have been executed
> (for some values of the variables of $M_1, \ldots, M_n$ not in M).

Horn clauses

The protocol is translated into a set of Horn clauses using four kinds of facts:

$attacker(M)$	the attacker may have M
$mess(M, M')$	the message M' may be sent on channel M
$begin(M)$	the event $\mathbf{begin}(M)$ may have been executed
$end(M)$	the event $\mathbf{end}(M)$ may have been executed

Proof technique

Secrecy: If $attacker(M)$ is not derivable (the verifier returns no clause in response to the query $attacker(M)$), then the attacker cannot have M (M is secret).

Proof technique

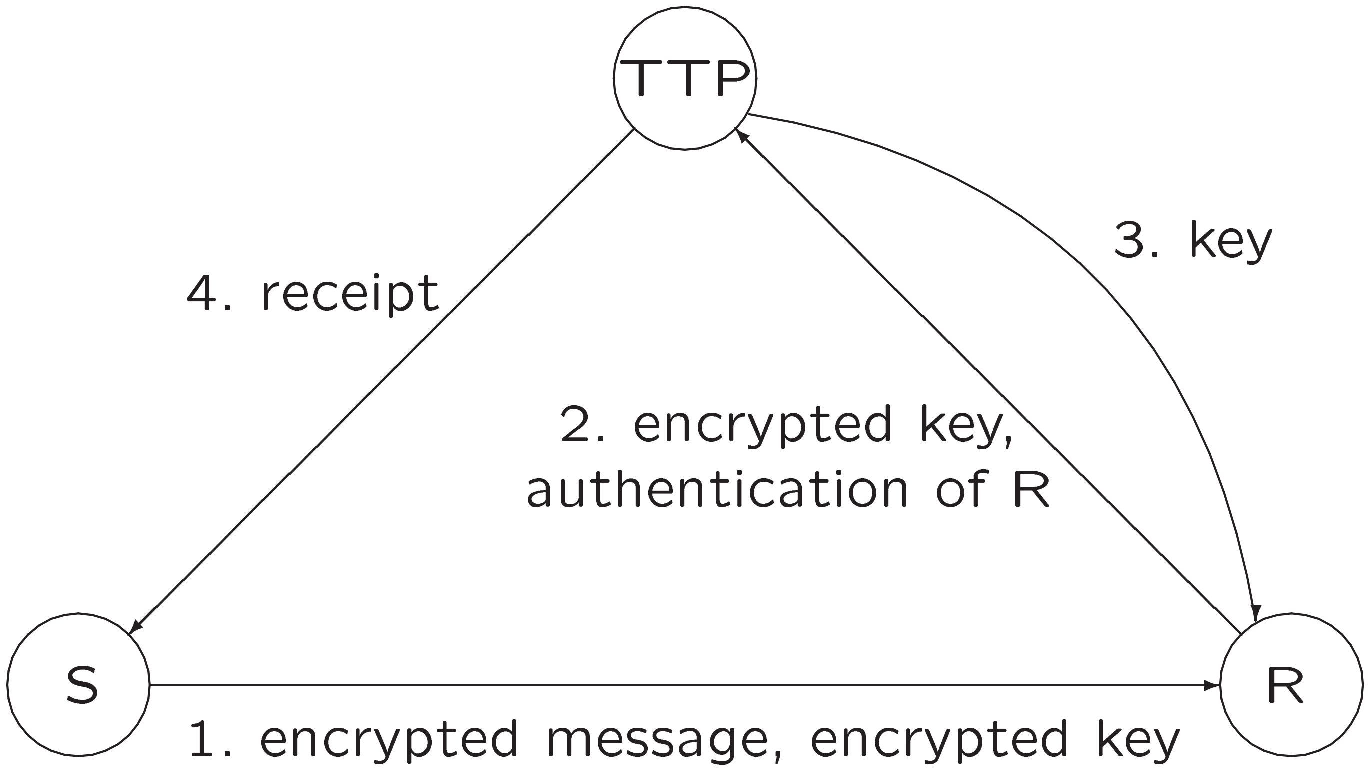
Secrecy: If $\mathit{attacker}(M)$ is not derivable (the verifier returns no clause in response to the query $\mathit{attacker}(M)$), then the attacker cannot have M (M is secret).

Correspondences: Given the query $\mathit{end}(M)$, assume that the verifier returns clauses that are all instances of

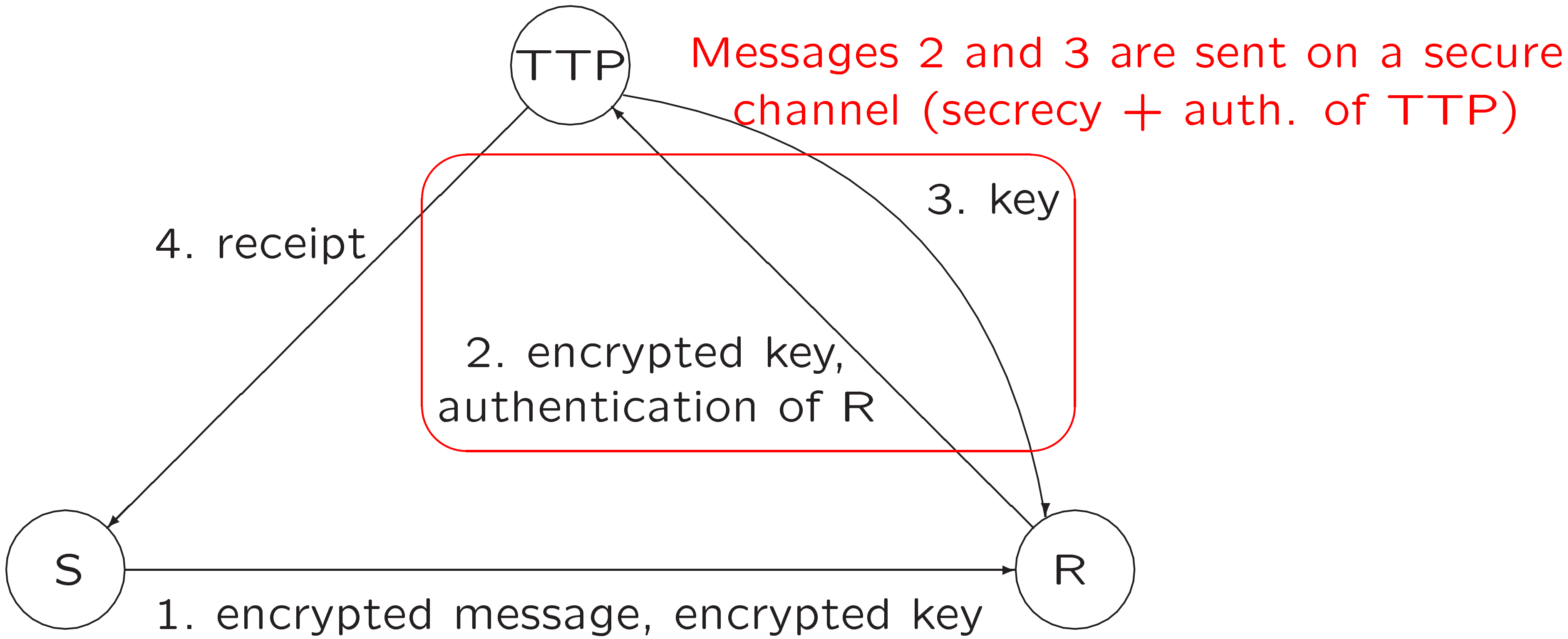
$$\mathit{begin}(M_1) \wedge \ldots \wedge \mathit{begin}(M_n) \wedge H \rightarrow \mathit{end}(M_0)$$

If $\mathbf{end}(\sigma M_0)$ is executed, for some substitution σ defined on the variables of M_0,
then we can extend σ such that $\mathbf{begin}(\sigma M_1)$, ..., $\mathbf{begin}(\sigma M_n)$ have been executed.

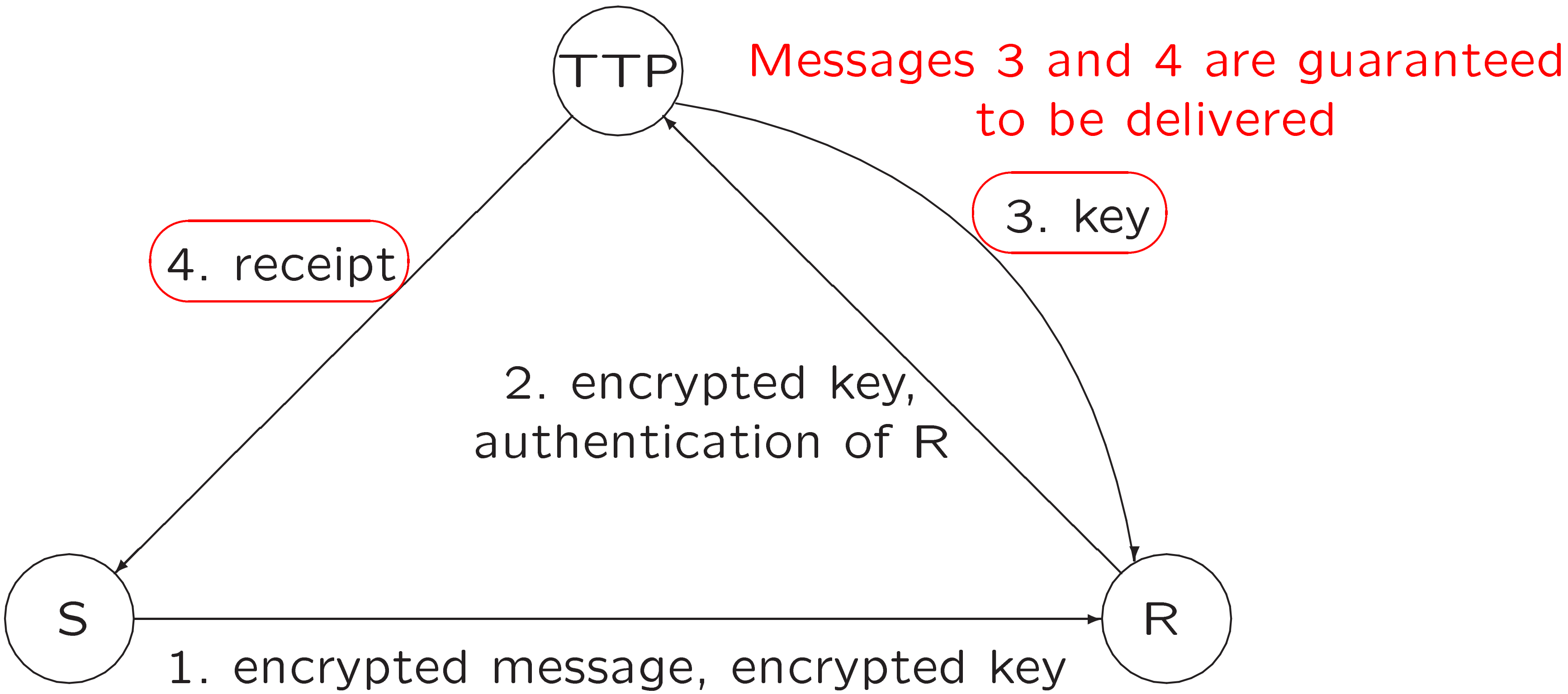
The certified email protocol



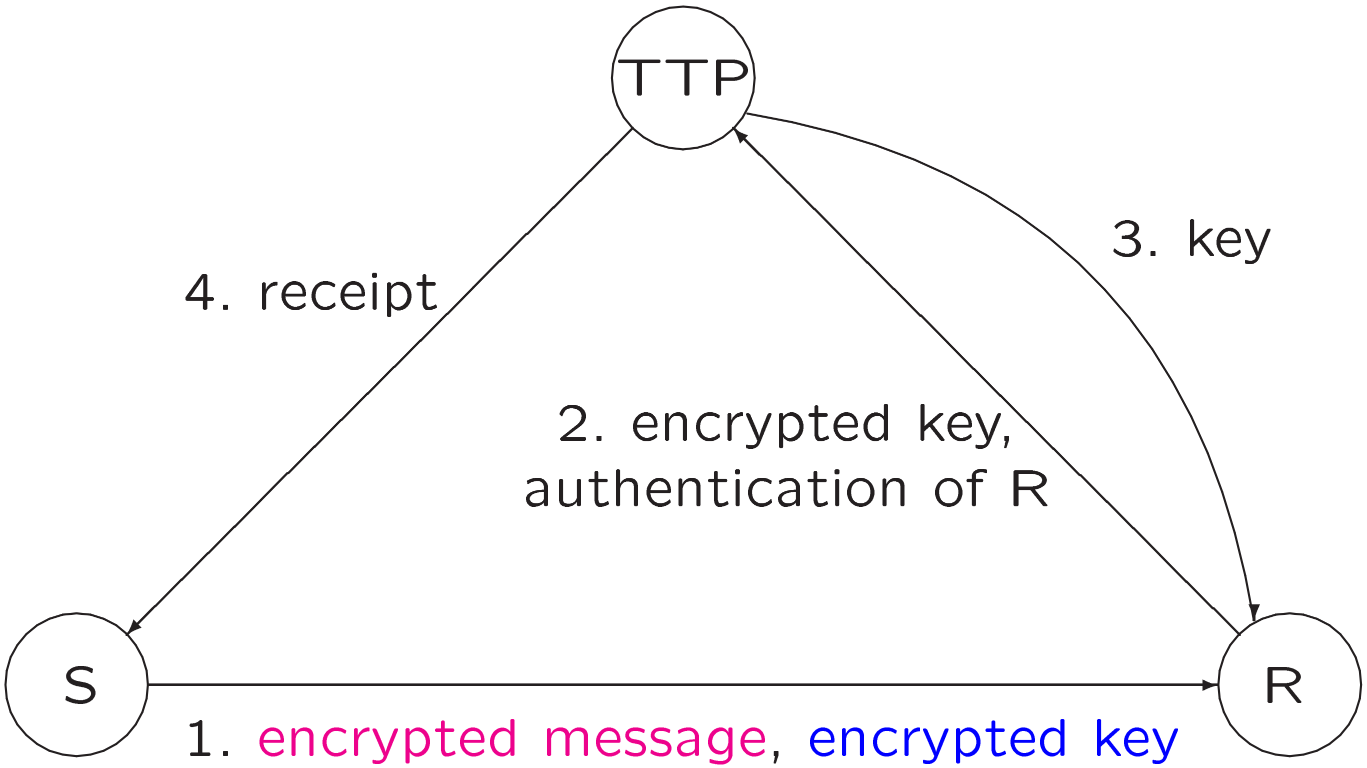
The certified email protocol



The certified email protocol



The protocol in more detail (1)

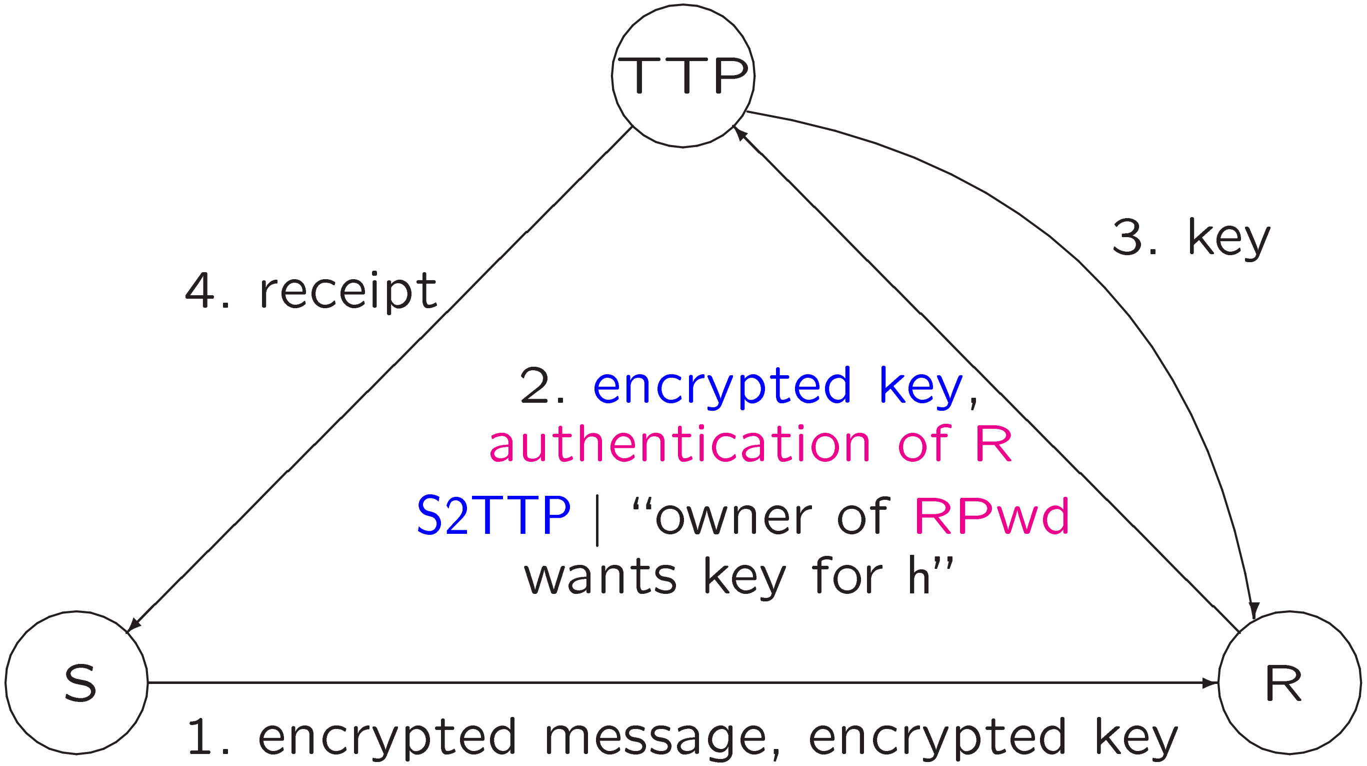


TTP | E(k, m) | cleartext | S2TTP

$h = H(\text{cleartext} \mid E(k, m))$

$S2TTP = A(TTPEncKey, S \mid \text{“give } k \text{ to R for h”})$

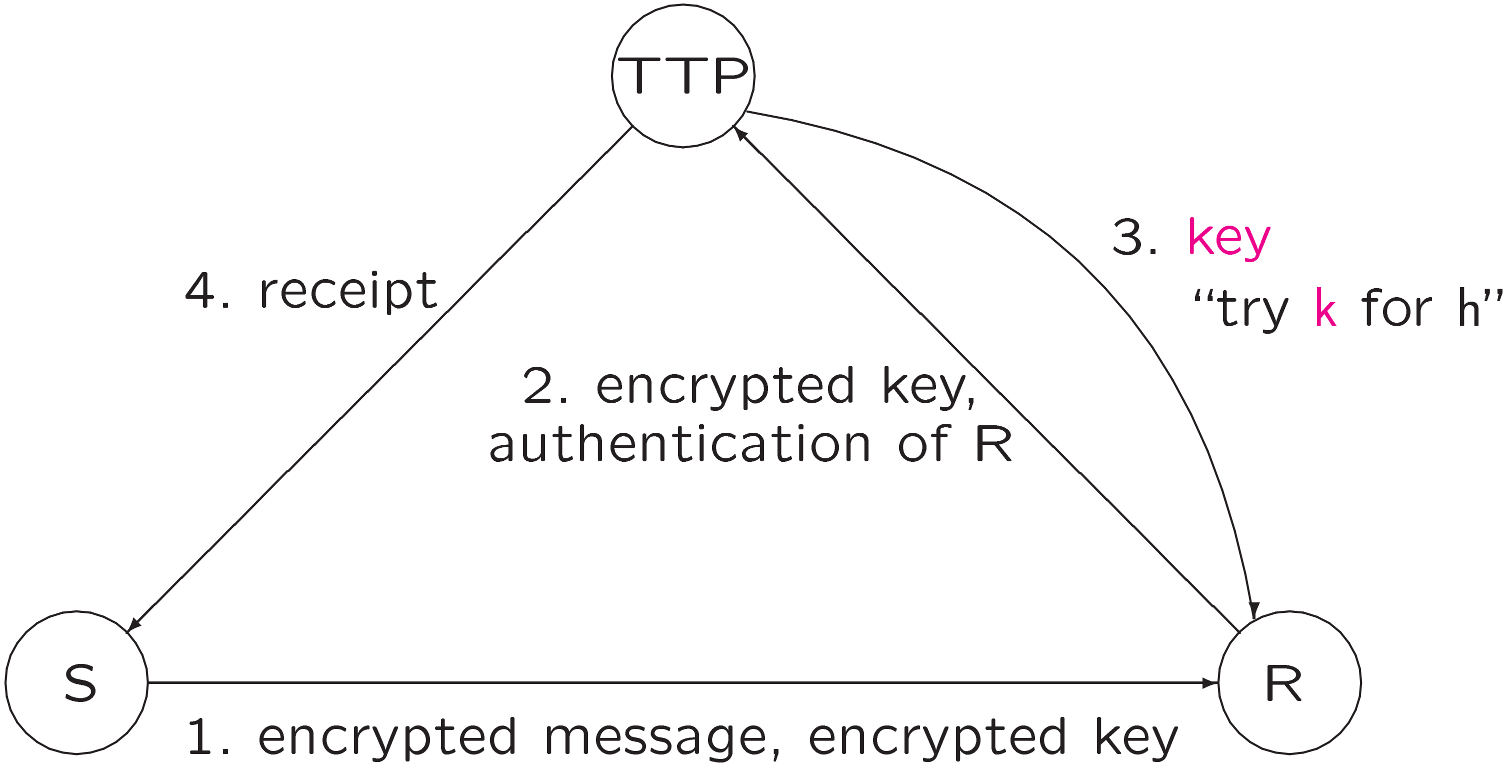
The protocol in more detail (2)



$$h = \mathsf{H}(\mathsf{cleartext} \mid \mathsf{E}(k, m))$$

$$\mathsf{S2TTP} = \mathsf{A}(\mathsf{TTPEncKey}, \mathsf{S} \mid \text{"give } k \text{ to R for h"})$$

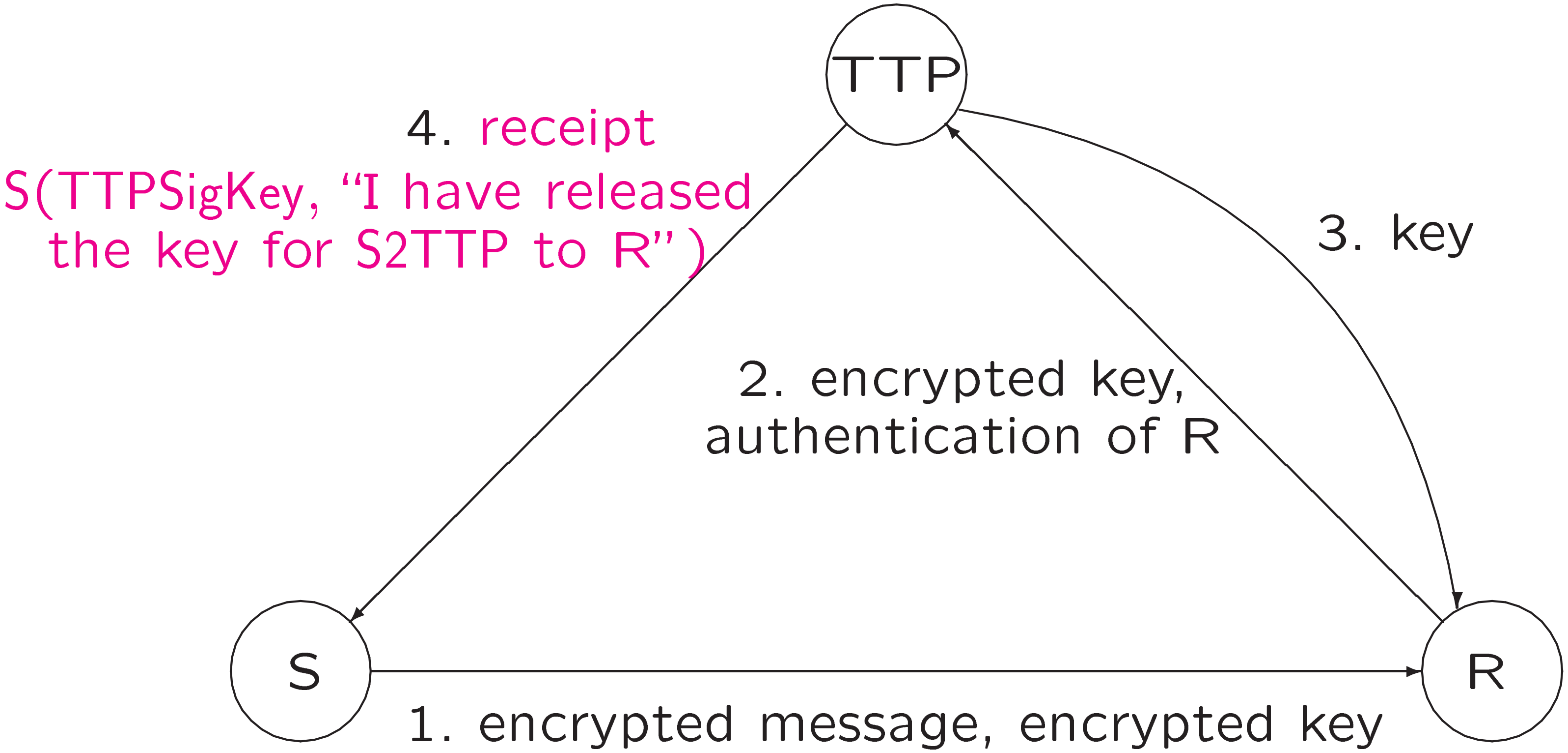
The protocol in more detail (3)



$\mathsf{h} = \mathsf{H}(\text{cleartext} \mid \mathsf{E}(k, m))$

$\mathsf{S2TTP} = \mathsf{A}(\mathsf{TTPEncKey}, \mathsf{S} \mid$ “give k to R for h” $)$

The protocol in more detail (4)



$h = H(\mathsf{cleartext} \mid E(k, m))$

$\mathsf{S2TTP} = A(\mathsf{TTPEncKey}, S \mid \text{“give k to R for h”})$

Code fragment

```
let processS =
  (* The attacker chooses possible recipients of the message *)
  in(c, recipient);
  (* Build the message to send *)
  new msgid;
  let m = Message(recipient,msgid) in
  (* Step 1 *)
  new k;
  let em = E(k,m) in
  let hs = H((cleartext, em)) in
  let S2TTP = A(TTPEncKey, (Sname, (Give, k, recipient, hs))) in

  out(recipient, (TTPname, em, cleartext, S2TTP));

  (* Step 4 *)
  !
  in(Sname, inmess4);
  let (=Released, =S2TTP, =recipient) = checkS(TTPVerKey, inmess4) in
  (* S knows that R has read the message *)

  0
  else out(Sname, inmess4).
```

Notes on the modelling

The protocol is assumed to be executed in the presence of a Dolev-Yao adversary that can intercept messages, compute, and send all messages it can build.

Dishonest participants (included in the adversary) can play the roles of the sender and/or the receiver.

Properties of the protocol: Secrecy

An adversary cannot have the messages m sent to R.

$$\mathsf{m} = \mathsf{Message}(\mathsf{Rname}, i)$$

This is easy to prove with our verifier:

– Give the query $attacker(\mathsf{Message}(\mathsf{Rname}, i))$.

– The verifier returns no clause, so the message is secret.

Properties of the protocol: Receipt

> The sender S has a receipt if and only if the receiver R has the message.

Furthermore, the receipt must be usable as a proof in front of judge.

The judge is a process that checks the receipt, and if the receipt is correct, executes an **end** event claiming "R has the message".

The property is then rephrased:

> The judge executes **end**(R has m) if and only if R has m.

Properties of the protocol: Receipt (continued)

The judge executes **end**(R has m) if and only if R has m.

This property is true only if:

- The messages from TTP to R and S are guaranteed to be delivered.
- The execution of the protocol is fair enough.
  (In particular, the process TTP executes all steps that it can execute).

However, our verifier cannot model these hypotheses !

Properties of the protocol: Receipt (continued)

So we split the proof of "if the event e_1 is executed then the event e_2 has or will be executed" into two parts:

- Automatic part: if the event e_1 is executed, then some events e'_2 have been executed (*before* e_1).

  This is a correspondence assertion that we can handle.

- Manual part: if the events e'_2 have been executed, then e_2 will be executed (*after* e'_2).

  This just consists in following a few execution steps after e'_2.

The automatic verifier does the difficult part: going backward into the execution history.

Example

Let us prove the implication:

If the event $\mathbf{end}(JudgeSays(\mathsf{Received}, \mathsf{Rname}, M_m))$ has been executed, then the event $\mathbf{end}(Rreceived(M_m))$ has or will be executed.

For this proof, S is included in the adversary, since he can behave dishonestly to try to prevent R from obtaining the message.

Example

Automatic part: If the event $\mathbf{end}(\mathit{JudgeSays}(\mathsf{Received}, \mathsf{Rname}, M_m))$ has been executed, then the events

$$\mathbf{begin}(\mathit{TTP\_send}((\mathsf{Rname}, M_{secchannel}), (\mathsf{Try}, M_k, M_{hr})))$$
$$\mathbf{begin}(\mathit{R\_has}(M_{secchannel}, \mathsf{E}(M_k, M_m), M_{hr}))$$

have been executed.

In response to the query $\mathit{end}(\mathit{JudgeSays}(\mathsf{Received}, \mathsf{Rname}, m))$, the verifier returns a clause of the form:

$$\begin{aligned} &\mathit{begin}(\mathit{TTP\_send}((\mathsf{Rname}, M'_{secchannel}), (\mathsf{Try}, k, M'_{hr}))) \\ &\wedge \mathit{begin}(\mathit{R\_has}(M'_{secchannel}, \mathsf{E}(k, m), M'_{hr})) \\ &\wedge H \rightarrow \mathit{end}(\mathit{JudgeSays}(\mathsf{Received}, \mathsf{Rname}, m)) \end{aligned}$$

which proves the correspondence assertion.

Example

Manual part: If the events

$$\mathbf{begin}(TTP\_send((\mathsf{Rname}, M_{secchannel}), (\mathsf{Try}, M_k, M_{hr})))$$
$$\mathbf{begin}(R\_has(M_{secchannel}, \mathsf{E}(M_k, M_m), M_{hr}))$$

have been executed, then $\mathbf{end}(Rreceived(M_m))$ will be executed.

Since TTP sends message 3 and its delivery is guaranteed, R is going to receive it and decrypt E(M_k, M_m) with the key M_k.

Demo

Conclusion

- We have verified the main security properties of the certified email protocol.

- Most of the time was spent in formally specifying the protocol in the pi calculus.

- We proved several versions of the protocol, with the secure channel established by three different means (including one using the SSH protocol).

  The protocol also supports 4 authentication modes of R.
  Total runtime of the tool: 2 min 30 s on an Intel Xeon 1.7 GHz.

- This study shows (and streched!) the power of the verifier.