Formally validated specification of a micro-payment protocol

P. Dargenton, D. Hirschkoff, P. Lescanne, E. Pommateau

Outline

1. sketch of a micro-payment protocol
2. its formalisation in Coq
3. proving *state integrity*

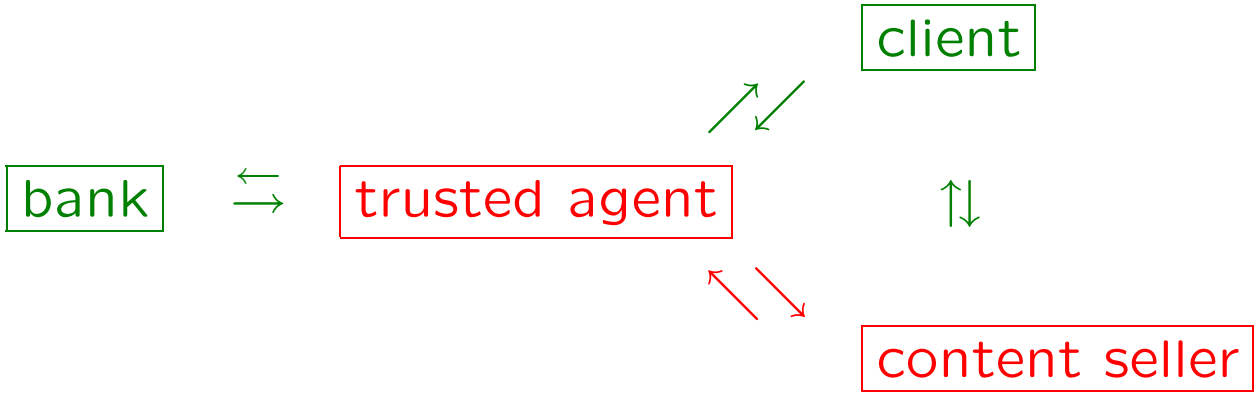
Origins of this work

- context: cooperation with industry
  startup developping micro-payment technology

Origins of this work

- context: cooperation with industry
  startup developping micro-payment technology

- micro-payment: scenario



Light Signatures

- client-server protocol
- many frequent and small transactions
  avoid heavy cryptography
- microtransactions: trade some security in favor of efficiency

A run of the protocol

- initialisation: client sends a (fresh) seed α
  both parties compute (and store) $H(\alpha), H(H(\alpha)), \ldots, H^{2N}(\alpha)$
  H: one-way hash function

A run of the protocol

- initialisation: client sends a (fresh) seed α
  both parties compute (and store) $H(\alpha), H(H(\alpha)), \ldots, H^{2N}(\alpha)$
  H: one-way hash function
- microtransactions
  - client's queries

(Clt_1) $Clt \rightarrow Srv : \langle Clt, 2 * I_{Clt}, \mathsf{Sign}(Q_{I_{Clt}}, 2 * I_{Clt}), Q_{I_{Clt}} \rangle$

  - server's answers

(Srv_1) $Srv \rightarrow Clt : \langle Srv, 2 * I_{Srv} + 1, \mathsf{Sign}(A_{I_{Srv}}, 2 * I_{Srv} + 1), A_{I_{Srv}} \rangle$

  - signature function: $\mathsf{Sign}(C, k) \stackrel{def}{=} H'(C, N_{2N-1-k})$

A run of the protocol

- initialisation: client sends a (fresh) seed α
  both parties compute (and store) $H(\alpha), H(H(\alpha)), \ldots, H^{2N}(\alpha)$
  H: one-way hash function
- microtransactions
  - client's queries

(Clt_1) $Clt \rightarrow Srv : \langle Clt, 2 * I_{Clt}, \mathsf{Sign}(Q_{I_{Clt}}, 2 * I_{Clt}), Q_{I_{Clt}} \rangle$

  - server's answers

(Srv_1) $Srv \rightarrow Clt : \langle Srv, 2 * I_{Srv} + 1, \mathsf{Sign}(A_{I_{Srv}}, 2 * I_{Srv} + 1), A_{I_{Srv}} \rangle$

  - signature function: $\mathsf{Sign}(C, k) \stackrel{def}{=} H'(C, N_{2N-1-k})$
- finishing the session

no more nonces $\rightarrow$ actual money transaction

Introducing time-outs

- possibility of message losses, local failures, attacks

- the light signatures protocol specifies the behaviour of both parties when an expected message does not arrive

... see paper

Mechanising the protocol:
traces and states

Paulson's approach

- use a general-purpose theorem prover to
  - specify the protocol
  - formally check some of its properties

- protocol $\leftrightarrow$ set of *traces*

- we use Coq instead of Isabelle

Generating traces

- traces are generated by and inductively defined relation

$$\frac{M \in \mathcal{T}}{\text{add } M' \text{ to } \mathcal{T}}$$

Generating traces

- traces are generated by and inductively defined relation

$$\frac{M \in \mathcal{T}}{\text{add } M' \text{ to } \mathcal{T}}$$

- such rules are used to describe
  - ▷ the agents' behaviour
  - ▷ the emission of messages forged by *the Spy*

Reasoning about traces

- traces give us a *global* view of protocol runs
  we can see everything is sent on the network

- use induction to prove theorems about the protocol

Reasoning about traces

- traces give us a *global* view of protocol runs
  we can see everything is sent on the network

- use induction to prove theorems about the protocol

- in the present work, no proof about properties of the protocol submitted to an attack (no Spy)
  - ▷ build a specification
  - ▷ proofs about the specification itself

Adding the agents' state

- each agent maintains a state:
  - ▷ current value of the index in the nonce sequence
  - ▷ current value of time-out counter

- rules of the form $\dfrac{E \;\;\&\;\; \mathcal{T}}{\boxed{\rightsquigarrow} \;\; E' \;\; ; \;\; \mathcal{T}'}$

Rules: an example

Rule c_1 : if $c < N$

$$\frac{\langle\!\langle\, c\,,\ \mathsf{TO}_C \parallel \_ \,\rangle\!\rangle \quad \& \quad \{\!\{ Srv,\ 2c+1,\ \mathsf{Sign}(A_c, 2c+1),\ A_c \}\!\}}{\boxed{\rightsquigarrow} \quad \{\!\{ Clt,\ 2(c+1),\ \mathsf{Sign}(Q_{c+1}, 2(c+1)),\ Q_{c+1} \}\!\} \ ; \ \langle\!\langle\, c+1\,,\ 0 \parallel \_ \,\rangle\!\rangle}$$

E & $\mathcal{T}$ $\leftrightarrow$ (micro e t)

```
c1 : (e:state)(t:trace)(micro e t) ->
  (c:nat)
  (in_set message t (msg Srv (S (mult (2) c))
                     (Sign (A c) (S (mult (2) c))) (A c)))
  /\ (state_c e)=c
  -> (micro (inc_c e) (add_set message
           (msg Clt (S (mult (2) c))
                    (Sign (Q (S c)) (S (mult (2) c))) (Q (S c)))   t))
 | ..
```

Rules: an example

Rule c_1 : if $c < N$

$$\frac{\langle\!\langle\, c\,,\ \mathsf{TO}_C \parallel \_ \,\rangle\!\rangle \quad \& \quad \{\!\{ Srv,\ 2c+1,\ \mathsf{Sign}(A_c, 2c+1),\ A_c \}\!\}}{\boxed{\rightsquigarrow} \quad \{\!\{ Clt,\ 2(c+1),\ \mathsf{Sign}(Q_{c+1}, 2(c+1)),\ Q_{c+1} \}\!\} \quad ; \quad \langle\!\langle\, c+1\,,\ 0 \parallel \_ \,\rangle\!\rangle}$$

E & $\mathcal{T}$ $\leftrightarrow$ `(micro e t)`

```
c1 : (e:state)(t:trace)(micro e t) ->
  (c:nat)
  (in_set message t (msg Srv (S (mult (2) c))
                     (Sign (A c) (S (mult (2) c))) (A c)))
  /\ (state_c e)=c
  -> (micro (inc_c e) (add_set message
           (msg Clt (S (mult (2) c))
                    (Sign (Q (S c)) (S (mult (2) c))) (Q (S c)))   t))
 | ..
```

$\rightarrow$ direct translation: gives confidence in the implementation

Local handling of state

- we did a simple extension to Paulson's approach

- internal state of each agent is handled *globally*
  - ▷ we are watching network traffic and all internal states ...
  - ▷ ... but we have a simple presentation

- we may want to check *state integrity*

  $\longrightarrow$ only a given agent can have an effet to its own state variables

Examining traces

The status of proofs

- in Coq, an hypothesis saying `(micro e t)` is a term having type `(micro e t)`

- difference w.r.t. Isabelle
  - ▷ dependent types
  - ▷ proofs are objects

i.e. traces contain their derivation

- we exploit this to establish state integrity

Dependently typed terms

lists

```
type list = nil :  list | cons :  int -> list -> list
[16;6;4]  ⟶   cons(16,cons(6,cons(4,nil)))
```

Dependently typed terms

lists

```
type list = nil :  list | cons :  int -> list -> list
[16;6;4]  ⟶   cons(16,cons(6,cons(4,nil)))
```

---

lists of length `n`

```
type list{nat} :  nil :  (list 0)
              | cons :  ∀ n. int -> (list n) -> (list (n+1))
[16;6;4]  ⟶  cons(2,16,cons(1,6,cons(0,4,nil)))
```

Proving state integrity

- given `H:(micro e' t)`, we define (`added_message` `H`), the last message sent "in `H`"

- and we prove

```
Theorem wf_rules : (e,e':state)(t:trace)
  (H:(micro e' t)) (next_state e H)
    ->
      (* either the client is sending, and s is invariant ... *)
   ( (state_s e)=(state_s e') /\ (msg_sender (added_message H))=Clt ) \/
      (* ... or the server is sending, and c is invariant.    *)
   ( (state_c e)=(state_c e') /\ (msg_sender (added_message H))=Srv ).
```

- `added_message`, `next_state` are defined by inspection of the structure of `H`

Computing on traces

- we could have done without dependent types to prove state integrity
  - ▷ show that $(\forall \mathtt{H})$, $(\mathtt{H} \rightarrow \mathtt{H'}) \Rightarrow \ldots$

Computing on traces

- we could have done without dependent types to prove state integrity
  - ▷ show that $(\forall \mathtt{H})$, $(\mathtt{H} \rightarrow \mathtt{H'}) \Rightarrow \ldots$
  - ▷ ad hoc formalism to analyse the shape of protocol rules

Computing on traces

- we could have done without dependent types to prove state integrity
  - ▷ show that $(\forall \mathtt{H})$, $(\mathtt{H} \rightarrow \mathtt{H'}) \Rightarrow \ldots$
  - ▷ ad hoc formalism to analyse the shape of protocol rules

- but the approach can be of interest
  think of using trace analysis e.g.:
  - ▷ to add hypotheses about the agents' behaviour or about protocol runs (e.g. fairness)

Computing on traces

- we could have done without dependent types to prove state integrity
  - ▷ show that $(\forall \mathtt{H})$, $(\mathtt{H} \rightarrow \mathtt{H'}) \Rightarrow \ldots$
  - ▷ ad hoc formalism to analyse the shape of protocol rules

- but the approach can be of interest
  think of using trace analysis e.g.:
  - ▷ to add hypotheses about the agents' behaviour or about protocol runs (e.g. fairness)
  - ▷ to make explicit causal relations between actions

Computing on traces

- we could have done without dependent types to prove state integrity
  - ▷ show that $(\forall \mathtt{H})$, $(\mathtt{H} \rightarrow \mathtt{H'}) \Rightarrow \ldots$
  - ▷ ad hoc formalism to analyse the shape of protocol rules

- but the approach can be of interest
  think of using trace analysis e.g.:
  - ▷ to add hypotheses about the agents' behaviour or about protocol runs (e.g. fairness)
  - ▷ to make explicit causal relations between actions

  $\rightarrow$ enrich the framework proposed by Paulson

Conclusions and future work

- a mechanised specification of a micro-payment protocol

  (and a report on the application of formal methods to an industrial product)

Conclusions and future work

- a mechanised specification of a micro-payment protocol

  (and a report on the application of formal methods to an industrial product)

- by adapting Paulson's approach to the Coq theorem prover, we have been able to do *more*

Conclusions and future work

- a mechanised specification of a micro-payment protocol

  (and a report on the application of formal methods to an industrial product)

- by adapting Paulson's approach to the Coq theorem prover, we have been able to do *more* in principle

dependent types allow us to reason on the shape of traces

Conclusions and future work

- a mechanised specification of a micro-payment protocol
  (and a report on the application of formal methods to an industrial product)

- by adapting Paulson's approach to the Coq theorem prover, we have been able to do *more* in principle

dependent types allow us to reason on the shape of traces

- future developments:
  - ▷ study the protocol under attackks
  - ▷ investigate applications of the methodology