Certification des protocoles cryptographiques

Slide 1

Frédéric Blanqui

École Polytechnique

http://www.lix.polytechnique.fr/~blanqui/

Plan

- Protocoles cryptographiques
- Approche inductive de Larry Paulson
- Exemple: Needham-Schroeder

Slide 2

Protocoles cryptographiques

Slide 3

objectifs: confidentialité, authentification, anonymat, etc.

exemple: Needham/Schroeder (1978) [17]

A et B veulent s'échanger deux nombres secrets tirés au hasard, Na et Nb, pour crypter leur communication

1. $A \to B : \{Na, A\}_{Kb}$
2. $B \to A : \{Na, Nb\}_{Ka}$
3. $A \to B : \{Nb\}_{Kb}$

problèmes: ce protocole assure-t-il que Na et Nb ne sont connus que de A et B? quand B reçoit le message 3, a-t-il été envoyé par A? etc.

Attaque de Lowe (1995) [13]

trouvée avec FDR

Slide 4

1. $A \to I : \{Na, A\}_{Ki}$
    1. $I(A) \to B : \{Na, A\}_{Kb}$
    2. $B \to A/I : \{Na, Nb\}_{Ka}$
2. $I \to A : \{Na, Nb\}_{Ka}$
3. $A \to I : \{Nb\}_{Ki}$
    3. $I(A) \to B : \{Nb\}_{Kb}$

Attaque de Lowe (1995) [14]

Slide 5

1. $A \to I : \{Na, A\}_{Ki}$
    1. $I(A) \to B : \{Na, A\}_{Kb}$
    2. $B \to A/I : \{Na, Nb\}_{Ka}$
2. $I \to A : \{Na, Nb\}_{Ka}$
3. $A \to I : \{Nb\}_{Ki}$
    3. $I(A) \to B : \{Nb\}_{Kb}$

correction:

1. $A \to B : \{Na, A\}_{Kb}$
2. $B \to A : \{Na, Nb, B\}_{Ka}$
3. $A \to B : \{Nb\}_{Kb}$

Une variante d'Otway-Rees [1]
(Abadi/Burrows/Needham, 1990)

Slide 6

1. $A \to B : Na, A, B, \{Na, A, B\}_{Ka}$
2. $B \to S : Na, A, B, \{Na, A, B\}_{Ka}, Nb, \{Na, A, B\}_{Kb}$
3. $S \to B : Na, \{Na, Kab\}_{Ka}, \{Nb, Kab\}_{Kb}$
4. $B \to A : Na, \{Na, Kab\}_{Ka}$

Attaque (Paulson, 1999) [19]

1. $A \to B/I : Na, A, B, \{Na, A, B\}_{Ka}$

   1. $I \to A : Ni, I, A, \{Ni, I, A\}_{Ki}$
   2. $A \to S/I : Ni, I, A, \{Ni, I, A\}_{Ki}, Na', \{Ni, I, A\}_{Ka}$
      $I(A) \to S : Ni, I, A, \{Ni, I, A\}_{Ki}, Na, \{Ni, I, A\}_{Ka}$
   3. $S \to A/I : Ni, \{Ni, Kia\}_{Ki}, \{Na, Kia\}_{Ka}$

Slide 7

4. $I(B) \to A : Na, \{Na, Kia\}_{Ka}$

dans le message
2. $B \to S : Na, A, B, \{Na, A, B\}_{Ka}, Nb, \{Na, A, B\}_{Kb}$
Nb n'est pas gardé secret

Conclusion

- même dans des protocoles en apparence simples, et en supposant les primitives cryptographiques <u>parfaites</u>, il peut y avoir des failles!
- la détection des failles est généralement indécidable [10, 9, 3, 21]

$\Rightarrow$ nécessité de développer des <u>théories</u> et des <u>outils</u> pour vérifier la sécurité des protocoles de manière formelle à l'aide d'une machine

Slide 8

Différentes approches

Slide 9

- model-checking/prouveurs spécialisés: [6, 16, 11, 8]
  + automatisation
  − pouvoir expressif limité, explosion combinatoire
  ⇒ simplification du modèle, limitation de la valeur des paramètres
- preuve formelle [4, 15, 20]
  + pouvoir expressif élevé
  − travail long et parfois fastidieux
- approche hybride [5]

Plan

Slide 10

- Protocoles cryptographiques
- Approche inductive de Larry Paulson [20]

voir `http://www.cl.cam.ac.uk/users/lcp/papers/protocols.html`
pour une biblio complète

- Exemple: Needham-Schroeder

Isabelle

voir `http://www.cl.cam.ac.uk/Research/HVG/Isabelle/`
et `http://www.cl.cam.ac.uk/users/lcp/papers/isabelle.html`

Slide 11

- assistant à la démonstration (Paulson, 1986)
  - générique: on peut y définir sa propre logique
  - méta-langage: logique d'ordre supérieur
  - tactiques: règles utilisateur + unification d'ordre supérieur
- développé à Cambridge (Paulson) et Münich (Nipkow)
- récente introduction (LNCS 2283) [18]

Protocoles certifiés dans Isabelle
(par Bella et Paulson)

voir `http://www.cl.cam.ac.uk/Research/HVG/Isabelle/library/HOL/Auth/`

Slide 12

- Needham-Schroeder symétrique
- Needham-Schroeder public
- Yahalom
- Otway-Rees
- Woo-Lam
- Recursive Authentication protocol
- Kerberos IV
- Transport Layer Security protocol (TLS 1.0 $\simeq$ SSL 3.0)
- Secure Electronic Transactions (SET)
- Zhou-Gollmann's non-repudiation protocol

Modélisation des messages

voir `http://www.cl.cam.ac.uk/Research/HVG/Isabelle/library/HOL/Auth/`

Slide 13

```
datatype msg   = Agent agent          (* identificateur *)
               | Number nat           (* texte clair *)
               | Nonce nat            (* nombre tiré au hasard *)
               | Key nat              (* clé *)
               | Hash msg             (* message haché *)
               | {msg,msg}            (* message composé *)
               | Crypt key msg        (* message crypté *)

datatype agent = Server               (* le gentil *)
               | Spy                  (* le méchant *)
               | Friend nat           (* les autres *)

datatype event = Says agent agent msg (* envoi *)
```

Conséquences

Slide 14

- les messages sont structurés et typés et non représentés comme une séquence de bits (pratique recommandée [2])
  - ⇒ pas de confusion possible entre les différents types de messages
  - ⇒ ne permet pas de certifier la correction contre les attaques exploitant l'absence de structure/typage (des études montrent cependant que la probabilité de cassage reste du même ordre [12])
- pas d'équivalence autre que syntaxique entre les messages
  - ⇒ ne permet pas de certifier la correction contre les attaques exploitant les relations algébriques entre messages cryptés [7]
    - XOR: `Crypt K (Crypt K X) = X`
    - RSA: `Crypt K1 (Crypt K2 X) = Crypt K2 (Crypt K1 X)`

Ensemble des sous-messages d'un ensemble de messages

```
consts parts ::  "msg set ⇒ msg set"

inductive "parts H" intros
Inj:  "X ∈ H ⟹ X ∈ parts H"
Fst:  "{X,Y} ∈ parts H ⟹ X ∈ parts H"
Snd:  "{X,Y} ∈ parts H ⟹ Y ∈ parts H"
Body:  "Crypt K X ∈ parts H ⟹ X ∈ parts H"
```

Slide 15

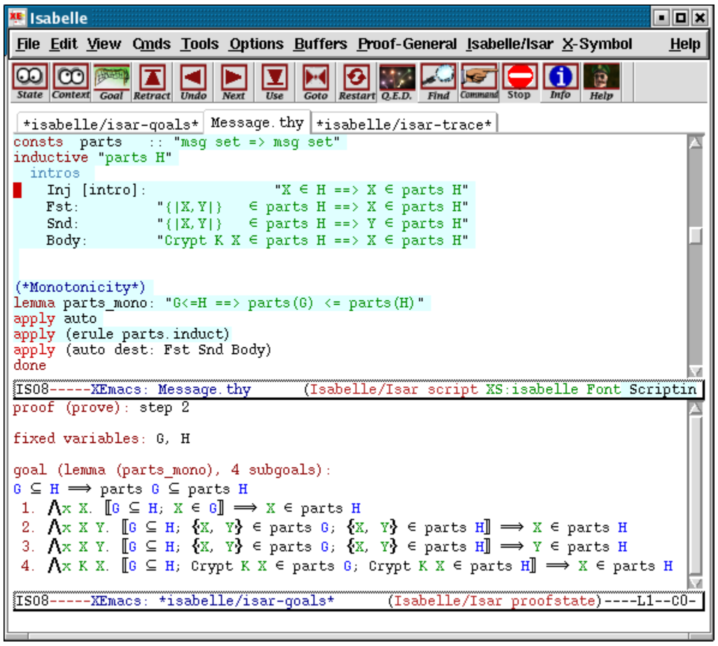
Rque: `X` n'est pas un sous-message de `Hash X`

Exemple de preuve dans Isabelle

```
(* monotonicity of parts *)
lemma parts_mono:  "G ⊆ H ⟹ parts(G) ⊆ parts(H)"
apply auto
apply (erule parts.induct)
apply (auto dest:  Fst Snd Body)
done
```

Slide 16

Slide 17



Slide 18

Ensemble des connaissances que l'on peut tirer d'un ensemble de messages

```
consts analz ::  "msg set ⇒ msg set"

inductive "analz H" intros
Inj:  "X ∈ H ⟹ X ∈ analz H"
Fst:  "{X,Y} ∈ analz H ⟹ X ∈ analz H"
Snd:  "{X,Y} ∈ analz H ⟹ Y ∈ analz H"
Decrypt:  "⟦Crypt K X ∈ analz H; Key(invKey K) ∈ analz H⟧ ⟹ X ∈ analz H"
```

<u>Rque:</u> suppose cryptographie parfaite (il faut avoir `invKey K` pour obtenir `X`)

⇒ correction forte: si un protocole admet une faille en supposant la cryptographie parfaite alors la faille existe réellement (l'inverse est bien sûr faux)

Ensemble des messages qui peuvent être synthétisés à partir d'un ensemble de messages

Slide 19

```
consts synth ::  "msg set ⇒ msg set"

inductive "synth H" intros
Inj:  "X ∈ H ⟹ X ∈ synth H"
Agent:  "Agent agt ∈ synth H"
Number:  "Number n ∈ synth H"
Hash :  "X ∈ synth H ⟹ Hash X ∈ synth H"
MPair:  "⟦X ∈ synth H; Y ∈ synth H⟧ ⟹ {X,Y} ∈ synth H"
Crypt:  "⟦X ∈ synth H; Key K ∈ H⟧ ⟹ Crypt K X ∈ synth H"
```

Rque: on ne peut pas synthétiser de nonces

⇒ un nombre tiré au hasard ne peut pas être deviné

Approche inductive de Larry Paulson [20]

Slide 20

- une trace est une séquence d'événements (envoi ou réception d'un message) en conformité avec le protocole ou synthétisé par l'espion
- un protocole peut être vu comme l'ensemble des traces possibles
- définir cet ensemble par induction donne un moyen de raisonner sur le protocole: par induction sur l'ensemble des traces

Formalisation commune à tous les protocoles

```
consts protocol :: "event list set"

inductive protocol intros
Nil:  "[] ∈ protocol" (* trace vide *)
Fake:  "⟦evs ∈ protocol; X ∈ synth(analz(spies evs))⟧
      ⟹ Says Spy B X # evs ∈ protocol" (* espion *)
```

Slide 21

```
consts spies :: "event list ⇒ msg set"

primrec
"spies [] = (Key`priK`bad) Un (Key`range pubK)" (* etat initial *)

"spies (Says A B X # evs) = insert X (spies evs)"
```

Plan

- Protocoles cryptographiques
- Approche inductive de Larry Paulson
- Exemple: Needham-Schroeder

Slide 22

Needham-Schroeder-Lowe

1. $A \rightarrow B : \{Na, A\}_{Kb}$

```
NS1:  "⟦evs ∈ ns; Nonce NA ∉ used evs⟧
⟹ Says A B (Crypt (pubK B) ⦃Nonce NA, Agent A⦄) # evs ∈ ns"
```

2. $B \rightarrow A : \{Na, Nb, B\}_{Ka}$

Slide 23

```
NS2:  "⟦evs ∈ ns; Nonce NB ∉ used evs;
Says A' B (Crypt (pubK B) ⦃Nonce NA, Agent A⦄) ∈ set evs⟧ ⟹
Says B A (Crypt (pubK A) ⦃Nonce NA, Nonce NB, Agent B⦄) # evs ∈ ns"
```

3. $A \rightarrow B : \{Nb\}_{Kb}$

```
NS3:  "⟦evs ∈ ns; Says A B (Crypt (pubK B) ⦃Nonce NA, Agent A⦄)
∈ set evs; Says B' A (Crypt (pubK A) ⦃Nonce NA, Nonce NB, Agent B)
∈ set evs⦄⟧ ⟹ Says A B (Crypt (pubK B) (Nonce NB)) # evs ∈ ns"
```

Correction

Slide 24

```
(* Theorems of the form X ∉ parts(spies evs) imply that NOBODY sends messages
containing X! *)

(* Spy never sees another agent's private key!  (unless it's bad at start) *)

lemma Spy_see_priK [simp]:  "evs ∈ ns ⟹ (Key (priK A) ∈ parts(spies evs)) =
(A ∈ bad)" by (erule ns.induct, auto)

lemma Spy_analz_priK [simp]:  "evs ∈ ns ⟹ (Key (priK A) ∈ analz(spies evs))
= (A ∈ bad)" by auto

(* Authenticity properties obtained from NS1 *)

(* It is impossible to re-use a nonce in both NS1 and NS2, provided the nonce
is secret.  (Honest users generate fresh nonces.)  *)

lemma no_nonce_NS1_NS2 [rule_format]:  "evs ∈ ns ⟹ Crypt (pubK C) ⦃NA',
Nonce NA, Agent D⦄ ∈ parts (spies evs) ⟶ Crypt (pubK B) ⦃Nonce NA, Agent A⦄
∈ parts (spies evs) ⟶ Nonce NA ∈ analz (spies evs)"
apply (erule ns.induct, simp_all)
by (blast intro: analz_insertI)+
```

```
(* Unicity for NS1: nonce NA identifies agents A and B *)
lemma unique_NA: "⟦Crypt(pubK B) {Nonce NA, Agent A } ∈ parts(spies evs);
Crypt(pubK B') {Nonce NA, Agent A'} ∈ parts(spies evs); Nonce NA ∉ analz
(spies evs); evs ∈ ns⟧ ⟹ A=A' ∧B=B'"
apply (erule rev_mp, erule rev_mp, erule rev_mp)
apply (erule ns.induct, simp_all)
by (blast intro: analz_insertI)+

(* Secrecy: Spy does not see the nonce sent in NS1 if A and B are secure *)
```

Slide 25

```
theorem Spy_not_see_NA: "⟦Says A B (Crypt(pubK B) {Nonce NA, Agent A}) ∈ set
evs; A ∉ bad; B ∉ bad; evs ∈ ns⟧ ⟹ Nonce NA ∉ analz (spies evs)"
apply (erule rev_mp)
apply (erule ns.induct, simp_all)
apply spy_analz
by (blast dest: unique_NA intro: no_nonce_NS1_NS2)+
```

```
(* Authentication for A: if she receives message 2 and has used NA to start a
run, then B has sent message 2. *)
lemma A_trusts_NS2_lemma [rule_format]: "⟦A ∉ bad; B ∉ bad; evs ∈ ns⟧ ⟹
Crypt (pubK A) {Nonce NA, Nonce NB, Agent B} ∈ parts (spies evs) ⟶ Says A B
(Crypt(pubK B) {Nonce NA, Agent A}) ∈ set evs ⟶ Says B A (Crypt(pubK A)
{Nonce NA, Nonce NB, Agent B}) ∈ set evs"
apply (erule ns.induct, simp_all)
by (blast dest: Spy_not_see_NA)+
```

Slide 26

```
theorem A_trusts_NS2: "⟦Says A B (Crypt(pubK B) {Nonce NA, Agent A}) ∈ set
evs; Says B' A (Crypt(pubK A) {Nonce NA, Nonce NB, Agent B}) ∈ set evs; A ∉
bad; B ∉ bad; evs ∈ ns⟧ ⟹ Says B A (Crypt(pubK A) {Nonce NA, Nonce NB,
Agent B}) ∈ set evs"
by (blast intro: A_trusts_NS2_lemma)

(* If the encrypted message appears then it originated with Alice in NS1 *)
lemma B_trusts_NS1 [rule_format]: "evs ∈ ns ⟹ Crypt (pubK B) {Nonce NA,
Agent A} ∈ parts (spies evs) ⟶ Nonce NA ∉ analz (spies evs) ⟶ Says A B
(Crypt (pubK B) {Nonce NA, Agent A}) ∈ set evs"
apply (erule ns.induct, simp_all)
by (blast intro!: analz_insertI)
```

```
(* Authenticity properties obtained from NS2 *)

(* Unicity for NS2:  nonce NB identifies nonce NA and agents A, B [unicity of
B makes Lowe's fix work] [proof closely follows that for unique_NA] *)

lemma unique_NB [dest]:  "⟦Crypt(pubK A) {|Nonce NA, Nonce NB, Agent B|} ∈
parts(spies evs); Crypt(pubK A') {|Nonce NA', Nonce NB, Agent B'|} ∈
parts(spies evs); Nonce NB ∉ analz (spies evs); evs ∈ ns⟧ ⟹ A=A' ∧NA=NA'
∧B=B'"
apply (erule rev_mp, erule rev_mp, erule rev_mp)
apply (erule ns.induct, simp_all)
by (blast intro:  analz_insertI)+
```

Slide 27

```
(* Secrecy:  Spy does not see the nonce sent in NS2 if A and B are secure *)

theorem Spy_not_see_NB [dest]:  "⟦Says B A (Crypt (pubK A) {|Nonce NA, Nonce
NB, Agent B|}) ∈ set evs; A ∉ bad; B ∉ bad; evs ∈ ns⟧ ⟹ Nonce NB ∉ analz
(spies evs)"
apply (erule rev_mp)
apply (erule ns.induct, simp_all)
apply spy_analz
by (blast intro:  no_nonce_NS1_NS2)+
```

```
(* Authentication for B: if he receives message 3 and has used NB in message
2, then A has sent message 3.  *)

lemma B_trusts_NS3_lemma [rule_format]:  "⟦A ∉ bad; B ∉ bad; evs ∈ ns⟧ ⟹
Crypt (pubK B) (Nonce NB) ∈ parts (spies evs) ⟶ Says B A (Crypt (pubK A)
{|Nonce NA, Nonce NB, Agent B|}) ∈ set evs ⟶ Says A B (Crypt (pubK B) (Nonce
NB)) ∈ set evs"
by (erule ns.induct, auto)

theorem B_trusts_NS3:  "⟦Says B A (Crypt (pubK A) {|Nonce NA, Nonce NB, Agent
B|}) ∈ set evs; Says A' B (Crypt (pubK B) (Nonce NB)) ∈ set evs; A ∉ bad; B ∉
bad; evs ∈ ns⟧ ⟹ Says A B (Crypt (pubK B) (Nonce NB)) ∈ set evs"
by (blast intro:  B_trusts_NS3_lemma)
```

Slide 28

```
(* Overall guarantee for B *)

(* If NS3 has been sent and the nonce NB agrees with the nonce B joined with
NA, then A initiated the run using NA.*)

theorem B_trusts_protocol:  "⟦A ∉ bad; B ∉ bad; evs ∈ ns⟧ ⟹ Crypt (pubK B)
(Nonce NB) ∈ parts (spies evs) ⟶ Says B A (Crypt (pubK A) {|Nonce NA, Nonce
NB, Agent B|}) ∈ set evs ⟶ Says A B (Crypt (pubK B) {|Nonce NA, Agent A|}) ∈
set evs" by (erule ns.induct, auto)
```

Quelques statistiques

Slide 29

fichier	nb lignes
Message	1060
Event	330
Keys	540
WooLam	170
NS_public	200
NS_shared	380
OtwayRees	460
Recur	500
Yahalom	640
TLS	910
KerberosIV	1570

References

[1] M. Abadi, M. Burrows, and R. Needham. A logic of authentication. *Proceedings of the Royal Society*, 426(1871):233–271, 1990.

[2] M. Abadi, M. Burrows, and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[3] R. Amadio and W. Charatonik. On name generation and set-based analysis in Dolev-Yao model. In *Proceedings of the 13th International Conference on Concurrency Theory*, Lecture Notes in Computer Science 2421, 2002.

[4] D. Bolignano. An approach to the formal verification of cryptographic protocols. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, 1996.

[5] D. Bolignano. Integrating proof-based and model-checking techniques for the formal verification of cryptographic protocols. In *Proceedings of the 10th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science 1427, 1998.

[6] E. Cohen. TAPS: a first-order verifier for cryptographic protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, 2000.

[7] H. Comon and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols.

[8] H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science*, ?(?):?–?, 2003.

[9] H. Comon, V. Cortier, and J. Mitchell. Tree automata with memory, set constraints and ping pong protocols. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 2076, 2001.

[10] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.

[11] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *Proceedings of the 15 IPDPS 2000 Workshops*, Lecture Notes in Computer Science 1800.

[12] J. Guttman, J. Thayer, and L. Zuck. The faithfulness of abstract protocol analysis: message authentication. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, 2001.

[13] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of the 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1055, 1996.

[14] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[15] J. Millen and H. Rueß. Local secrecy for stated-based models. In *Workshop on Formal Methods in Computer Security*, 2000.

[16] D. Monniaux. Abstracting cryptographic protocols with tree automata. In *Proceedings of the 6th International Symposium on Static Analysis*, Lecture Notes in Computer Science 1694, 1999.

[17] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–998, 1978.

[18] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. `http://www.cl.cam.ac.uk/Research/HVG/Isabelle/`.

[19] L. Paulson. Proving security protocols correct. In *Proceedings of the 14th IEEE Symposium on Logic in Computer Science*, 1999.

[20] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1/2):85–128, 1998.

[21] M. Rusinowitch. Protocol insecurity with finite number of sessions is np-complete. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, 2001.