Action Spécifique Sécurité

Bruno Blanchet
Équipe Interprétation abstraite
sous la direction de Patrick Cousot
Département d'Informatique,
École Normale Supérieure

Mai 2002

Partie I : le travail de l'équipe Interprétation abstraite du LIENS pour l'Action Spécifique Sécurité

Analyse de protocoles cryptographiques
(David Monniaux, Bruno Blanchet)

Analyse de confidentialité pour le code mobile
(Jérôme Féret)

Tatouage sémantique de logiciel
(Patrick Cousot, projet TUAMOTU du RNRT)

Sécurisation de programmes par contrôleur d'exécution
(Patrick Cousot, avec Radhia Cousot, École Polytechnique)

Futur : Analyse de programmes probabilistes pour la sécurité
(David Monniaux)

Analyse de protocoles cryptographiques

Procédures de décision pour les logiques de croyances, comme BAN et GNY (David Monniaux).

Vérification de secret en calculant une surapproximation de l'ensemble des messages que l'attaquant peut avoir, représentée par des automates d'arbres (David Monniaux).

Représentation abstraite des protocoles par des clauses de Horn (Bruno Blanchet)

Analyse de confidentialité pour le code mobile (Jérôme Feret)

Analyse de processus du pi calcul et des ambients pour obtenir une description des communications dans un système, et compter les occurrences de processus.

Peut traiter un nombre illimité d'instances.

Distingue différentes instances du même processus.

$\Rightarrow$ Meilleure précision que les analyses précédentes.

Rapide en pratique (moins d'1 s sur beaucoup d'exemples).

Tatouage sémantique de logiciel (Patrick Cousot)

Tatouage = marque indélébile qui indique l'origine du logiciel (protection contre le piratage).

Idée : l'interprétation abstraite permet de déterminer des propriétés sémantiques des programmes

$\Rightarrow$ construire un tatouage sémantique.

Tatouage invisible à l'exécution,
mais récupérable par des analyseurs abstraits.

Technique brevetée avec Thalès.

Sécurisation de programmes par contrôleur d'exécution (P. et R. Cousot)

But : Limiter les comportements d'un programme P à ceux autorisés par une spécification donnée.

La spécification est exprimée par un programme M (contrôleur d'exécution), qui peut avoir tous les comportements autorisés.

On transforme P en un programme qui bloque quand il allait exécuter une action interdite par M.

Analyse de programmes probabilistes (David Monniaux)

But : trouver un majorant de la probabilité d'un certain événement (par exemple une erreur, une panne, ...)

Applications à la sécurité, à explorer:

- Vérification de protocoles cryptographiques tenant compte des aspects probabilistes.

- Modélisation probabiliste de l'environnement (réseau par exemple)

Part II: Secrecy for cryptographic protocols (Bruno Blanchet)

Introduction

Protocol representation

Solving algorithm

Results and conclusion

Example

Denning-Sacco key distribution protocol (simplified)

Message 1.	$A \rightarrow B:$	$\{\{k\}_{sk_A}\}_{pk_B}$	k fresh
Message 2.	$B \rightarrow A:$	$\{s\}_k$	

The goal of the protocol is to keep s secret.

The attack

The (well-known) attack against this protocol.

Message 1.	$A \rightarrow C$:	$\{\{k\}_{sk_A}\}_{pk_C}$
Message 1'.	$C(A) \rightarrow B$:	$\{\{k\}_{sk_A}\}_{pk_B}$
Message 2.	$B \rightarrow C(A)$:	$\{s\}_k$

The attacker C impersonates A and obtains the secret s.

Our goal

Goal: a verifier for cryptographic protocols

- Fully automatic.
- For an unbounded the number of sessions of the protocol.
- Efficient (no state space explosion).

Example: secrecy properties of Skeme in 17 ms, less than 2 Mb.

Our solution

Two ideas (extending [Weidenbach, CADE'99]):

- a simple abstract representation of these protocols, by a set of Horn clauses (logic programming rules);

- an efficient solving algorithm to find which facts can be derived from these clauses.

Using this, we can prove secrecy properties of protocols,
or exhibit attacks showing why a message is not secret.

For the moment, we handle shared- and public-key cryptography, hash functions, Diffie-Hellman key agreements.

Protocol representation

Protocol representation

- Messages $\rightsquigarrow$ terms

  $M ::= x \mid f(M_1, \ldots, M_n) \mid k[M_1, \ldots, M_n]$

  $\mathsf{pencrypt}(c_0, \mathsf{pk}(sk_A))$.

- Properties $\rightsquigarrow$ facts

  $F ::= \mathsf{attacker}(M)$.

- Protocol, attacker $\rightsquigarrow$ Horn clauses

  $F_1 \wedge \ldots \wedge F_n \rightarrow F$

  $\mathsf{attacker}(m) \wedge \mathsf{attacker}(pk) \rightarrow \mathsf{attacker}(\mathsf{pencrypt}(m, pk))$.

Example - Cryptographic primitives

- Encryption $\mathsf{pencrypt}(m, pk)$.
  $\mathsf{attacker}(m) \wedge \mathsf{attacker}(pk) \longrightarrow \mathsf{attacker}(\mathsf{pencrypt}(m, pk))$

- Public key generation $\mathsf{pk}(sk)$.
  $\mathsf{attacker}(sk) \longrightarrow \mathsf{attacker}(\mathsf{pk}(sk))$

- Decryption $\mathsf{pdecrypt}(\mathsf{pencrypt}(m, \mathsf{pk}(sk)), sk) = m$.
  $\mathsf{attacker}(\mathsf{pencrypt}(m, \mathsf{pk}(sk))) \wedge \mathsf{attacker}(sk) \longrightarrow \mathsf{attacker}(m)$

General treatment of primitives

- Constructors $f(M_1, \ldots, M_n)$
  $\mathsf{attacker}(x_1) \wedge \ldots \wedge \mathsf{attacker}(x_n) \rightarrow \mathsf{attacker}(f(x_1, \ldots, x_n))$

- Destructors $g(M_1, \ldots, M_n) = M$
  $\mathsf{attacker}(M_1) \wedge \ldots \wedge \mathsf{attacker}(M_n) \rightarrow \mathsf{attacker}(M)$

  (There may be several equations defining a function.)

Names

Normally, fresh names are created each time the protocol is run. Here, we only distinguish two names when they are created after receiving different messages.

Each name k becomes a function of the messages previously received:

$$k[M_1, \ldots, M_n].$$

(Skolemisation)

These functions can only be applied by the principal that creates the name, not by the attacker.

Denning-Sacco protocol

- $A \rightarrow B : \{\{k\}_{sk_A}\}_{pk_B}$ $\quad$ k fresh

  A talks with any principal represented by its public key $\mathsf{pk}(x)$.
  A sends to it the message $\{\{k\}_{sk_A}\}_{\mathsf{pk}(x)}$.

  $\mathsf{attacker}(\mathsf{pk}(x)) \rightarrow \mathsf{attacker}(\mathsf{pencrypt}(\mathsf{sign}(k[\mathsf{pk}(x)], sk_A[]), \mathsf{pk}(x)))$.

- $B \rightarrow A : \{s\}_k$

  B has received a message $\{\{y\}_{sk_A}\}_{pk_B}$.
  B sends $\{s\}_y$.

  $\mathsf{attacker}(\mathsf{pencrypt}(\mathsf{sign}(y, sk_A[]), \mathsf{pk}(sk_B[]))) \rightarrow \mathsf{attacker}(\mathsf{sencrypt}(\mathsf{s}, y))$.

General coding of a protocol

If a principal A has received the messages $M_1, \ldots, M_n$ and sends the message M,

$$\mathsf{attacker}(M_1) \wedge \ldots \wedge \mathsf{attacker}(M_n) \longrightarrow \mathsf{attacker}(M).$$

Approximations

- The freshness of nonces is partially modeled.
- The number of times a message appears is ignored,
  only the fact that is has appeared is taken into account.
- The state of the principals is not fully modeled.

These approximations are keys for an efficient verification.
Solve the state space explosion problem.
No limit on the number of runs of the protocols.
$\Rightarrow$ essential for the certification of protocols.

Secrecy

Secrecy criterion:

If attacker(M) cannot be derived from the rules, then M is secret.

The term M cannot be built by an attacker.
(This is a weaker notion of secrecy than non-interference.)

The solving algorithm will determine whether a given fact can be derived from the rules.

Solving algorithm

First phase: completion by resolution with selection

Selection function $sel(F_1 \wedge \ldots \wedge F_n \rightarrow F) \subseteq \{F_1, \ldots, F_n\}$.

$$sel(F_1 \wedge \ldots \wedge F_n \rightarrow F) = \begin{cases} \emptyset \text{ if } \forall i \in \{1, \ldots, n\}, F_i = \text{attacker}(x) \\ \{F_i\} \text{ different from attacker}(x), \\ \qquad \text{of maximal size, otherwise} \end{cases}$$

Resolution

$$\frac{R = F_1 \wedge \ldots \wedge F_n \rightarrow F \qquad R' = F'_1 \wedge \ldots \wedge F'_{n'} \rightarrow F'}{\sigma F_1 \wedge \ldots \wedge \sigma F_n \wedge \sigma F'_2 \wedge \ldots \wedge \sigma F'_{n'} \rightarrow \sigma F'}$$

where σ is the most general unifier of F and F'_1,
$sel(R) = \emptyset$, and $F'_1 \in sel(R')$.

First phase

Perform resolution steps until a fixed point is reached, take only the rules R with $sel(R) = \emptyset$.

Example of a step:

$$\frac{\begin{array}{l}\mathsf{attacker}(x) \wedge \mathsf{attacker}(y) \longrightarrow \mathsf{attacker}(\mathsf{pencrypt}(x, y)) \\ \mathsf{attacker}(\mathsf{pencrypt}(\mathsf{sign}(z, sk_A[]), \mathsf{pk}(sk_B[]))) \longrightarrow \mathsf{attacker}(\mathsf{sencrypt}(\mathsf{s}, z))\end{array}}{\mathsf{attacker}(\mathsf{sign}(z, sk_A[])) \wedge \mathsf{attacker}(\mathsf{pk}(sk_B[])) \longrightarrow \mathsf{attacker}(\mathsf{sencrypt}(\mathsf{s}, z))}$$

Main theorem:

The rules obtained after the first phase prove the same facts as the starting rules.

Why it works

$$\frac{R = F_1 \wedge \ldots \wedge F_n \rightarrow F \qquad R' = F'_1 \wedge \ldots \wedge F'_{n'} \rightarrow F'}{\sigma F_1 \wedge \ldots \wedge \sigma F_n \wedge \sigma F'_2 \wedge \ldots \wedge \sigma F'_{n'} \rightarrow \sigma F'}$$

The conditions "$sel(R) = \emptyset$ and $sel(R') = \{F'_1\}$" strongly restrict the possible unifications

Essentially, two non-variable terms must unify for a new rule to be created.

Second phase

Second phase: backward depth-first search.

This is the usual Prolog algorithm, except that we detect cycles.

Technique similar to the ordered resolution with selection [Weidenbach, CADE'99], but without ordering.

Results and conclusion

Experimental results

Pentium III 1 GHz.

Protocol	Result	ms
Needham-Schroeder public key	Attack [Lowe]	8
Needham-Schroeder public key corrected	Secure	5
Needham-Schroeder shared key	Attack [Denning]	35
Needham-Schroeder shared key corrected	Secure	49
Denning-Sacco	Attack [AN]	5
Denning-Sacco corrected	Secure	4
Otway-Rees	Secure	13
Otway-Rees, variant of Paulson98	Attack [Paulson]	14
Yahalom	Secure	8
Simpler Yahalom	Secure	16
Main mode of Skeme	Secure	17

Conclusion

- A particularly efficient verifier
- Can handle complex protocols
- No limit on the number of runs of the protocol
- Can be used for certification of protocols

Other work already done

- Extend this to prove authenticity.

- Link with a linear logic model of cryptographic protocols.

  Our representation is an abstraction of the linear logic model. (We forget how many times each message is repeated.)

- Link with typing with Martín Abadi.

  Our representation is equivalent to the best instance of a very general type system for extensions of the pi calculus with cryptographic primitives. (It proves exactly all secrecy properties provable in any instance of the type system.)

Future work

- Termination.
  Find an interesting subclass of protocols on which the algorithm terminates.

- Other cryptographic primitives (xor, ...).

- Other properties (non-interference, equivalences of processes)

- Proof of implementations of protocols, not specifications