

Crypto Verif : A computationally sound Mechanized Prover for security protocols

SUMMARY

Authors: Daniel Kolokosso, Patrice Cogne, Frederic Mejean, Jérémie Tharaud , Loïc Pillard

November 20, 2009

CryptoVerif has been computed by Bruno Blanchet of the CNRS, Paris. It's a new mechanized prover of security protocols. Its particularity is that it does not rely on the Dolev Yao model, but on the computational model (cryptographic primitives are materialised by black boxes). The adversary is a probabilistic polynomial time Turing Machine. Games are represented in a process calculus inspired by the pi-calculus.

this proof algorithm consists on game transformations. The initial game is the protocol to prove, and the last game is the ideal protocol. The transformations are justified by observational equivalence represented $=_v$ between two processes (respecting invariant1*,invariant 2*,invariant3 *rules), with V a set of variables.

Intuitively, observational equivalence between Q and Q' means that an adversary represented by the context C, cannot distinguish either process Q or Q' has been choosed, from the outside. So this equivalence allow us to build game transformations, in order to prove the secrecy of a protocol.

For this achievement, some secrecy criterias are defined in this paper : preservation of secrecy and one-session secrecy. These definitions have been adapted to this study, and are applied like following:

let Q_0 be a protocol to analyse, we apply the syntactic tools and security primitives transformations to get $Q'_0 =_v Q_0$, and if Q'_0 preserve secrecy, then Q_0 also preserve the secrecy.

Security primitives are cryptographic functions like "encryption", "mac" (message authentication code),... There are some predefined transformations related to security primitives, which the program try to execute. When it doesn't succeed it applies syntactic transformations, and then retry.

Syntactic transformations are :

- Remove-Assign : when x is defined by an assignment like $\text{let } x[i\dots] : T = M$, we replace x with its value.
- SaRename: renaming variables such that each variable has a single definition.

- Simplify , which simplifies the process.

The tools are used according to a proof strategy which determine the order, the combinaison of the transformations to apply. The program ends when it succeeds on proving a secrecy, or when it has tried with no succes all the possible transformations.

* invariant 1 = single definition (each variable appear at most 1 time in each branch of the calculus)
invariant 2 = defined variables (each variable must be initialize before to be used)
invariant 3 = well-typed