Presentation

Master thesis of Léo Ducas,
supervised by Mathieu Baudet (ANSSI).

Conception of a language for cryptographic reductions

Cryptographic reduction

A cryptographic reduction transform

an attacker against a cryptographic construction

into

a solver of some believed hard problem

Exemple :

An attacker on the *Cramer-Shoup* encryption can be transformed into an algorithm solving the *Diffie-Hellman problem.*

Reliability of proofs

Cryptographic reductions deals with many probabilistic algorithms with complex interactions

Mistakes in security proofs are possible !

- Ex : OAEP Scheme [Bellar & Rogaway, 1994]

Formal proofs

- More reliable
- May be assited / automatisable

But also

- Logical and pedagogical interest

Existing formal frameworks for cryptographic proofs

- *CryptoVerif Tool* [Blanchet, 2006]
  Concrete security, game-based proofs, automatised
- Pseudo-code of Backes et al. [Backes et al., 2008]
  asymptotic security, game-based proofs, assisted by *Isabelle/HOL*
- *The computational SLR* [Yu Zhang, 2009]
  asymptotic security, game-based proofs, manual
- *Framework for language-based cryptographic proofs* [Barthe et al., 2009]
  Concrete security, game-based proofs, assisted by *Coq*

Our Approach

Constructive approach, with explicit reductions

As suggested by P. Rogaway [Rogaway, 2006]

3 steps to prove security :

1/ Explicitly write reductions

2/ Prove its correctness

3/ Prove its efficiencity (concrete or asymptotic)

Our work focuses on step 1/

Goals

Conception of a language for cryptographic reductions

- Complete enough describe modern cryptographic concept and state corresponding security results
- Simple enough to allow futures formals proofs on the programs written in this language
- Based on Lambda-Calculus (higher order)
- With polymorphic typing (a posteriori)

Summary

1. Introduction

2. The langage
    - Higher order in cryptography
    - Lambda-Calculus « à la Moggi »
    - Implémentation examples

3. Algebraic models
    - Presentation of algebraic (or generic) models
    - Taking advantage of polymorphism

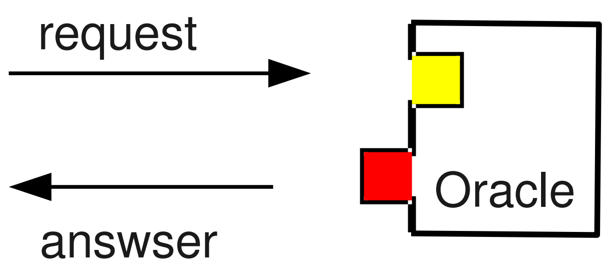
4. Conclusion
    - Results
    - Other problems
    - Bibliography

Higher order in Cryptography

Oracles are used to modelize information the attacker can get

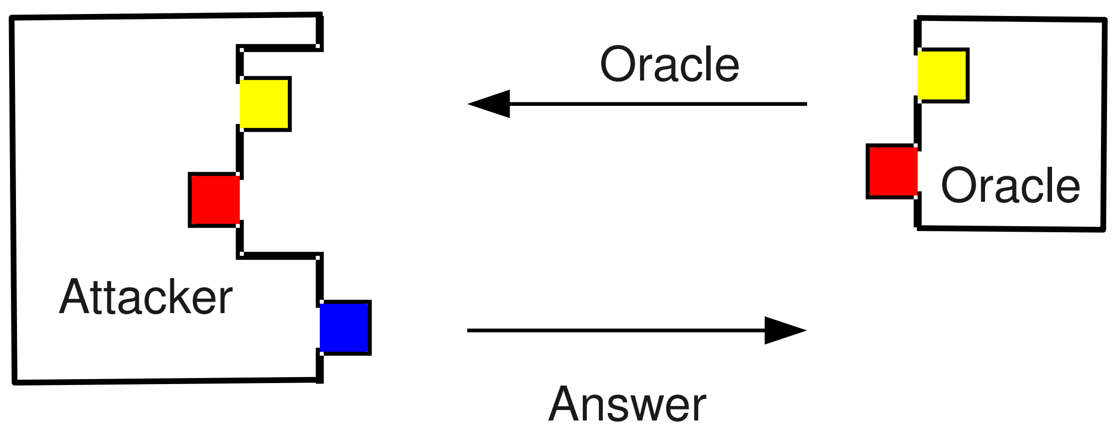
Ex : (Signature scheme) the attacker may know many signed messages.

In the worst case, he can choose those messages.



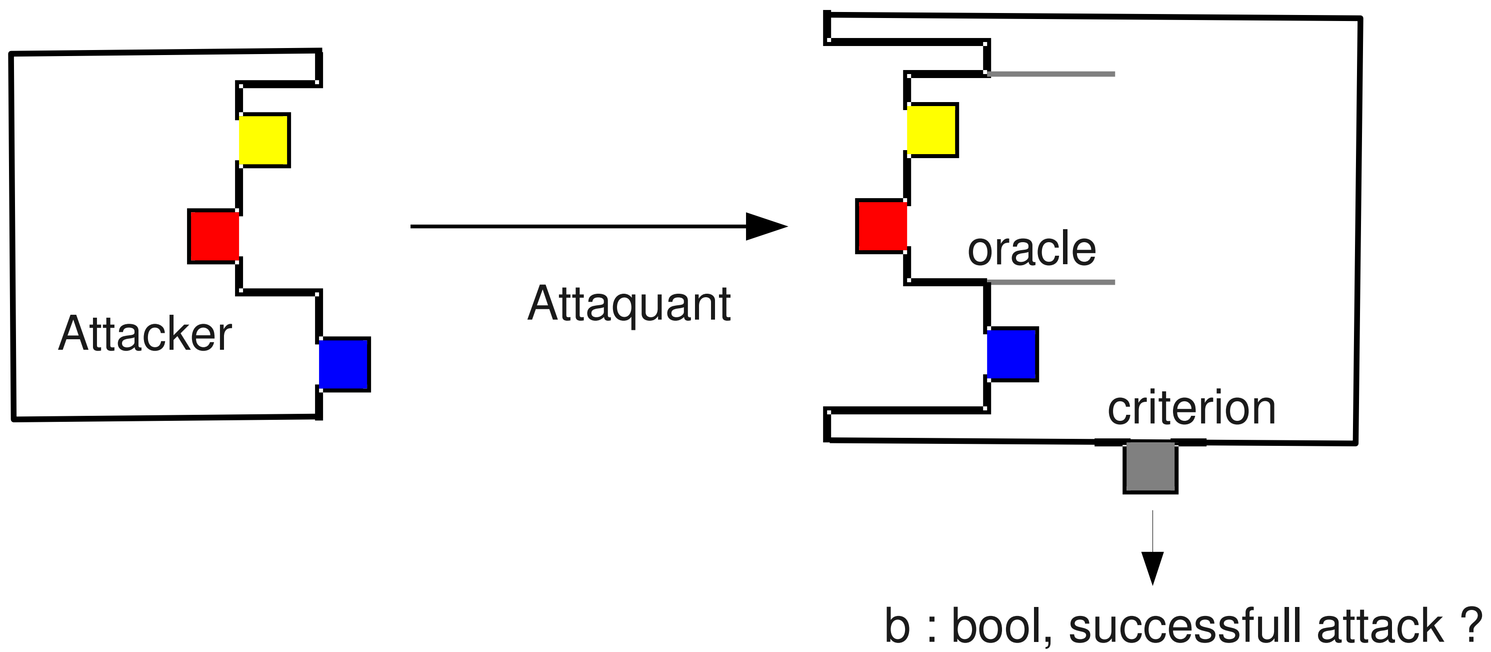
Oracle : Request $\rightarrow$ answer (Ordre 1)

Higher order in Cryptography



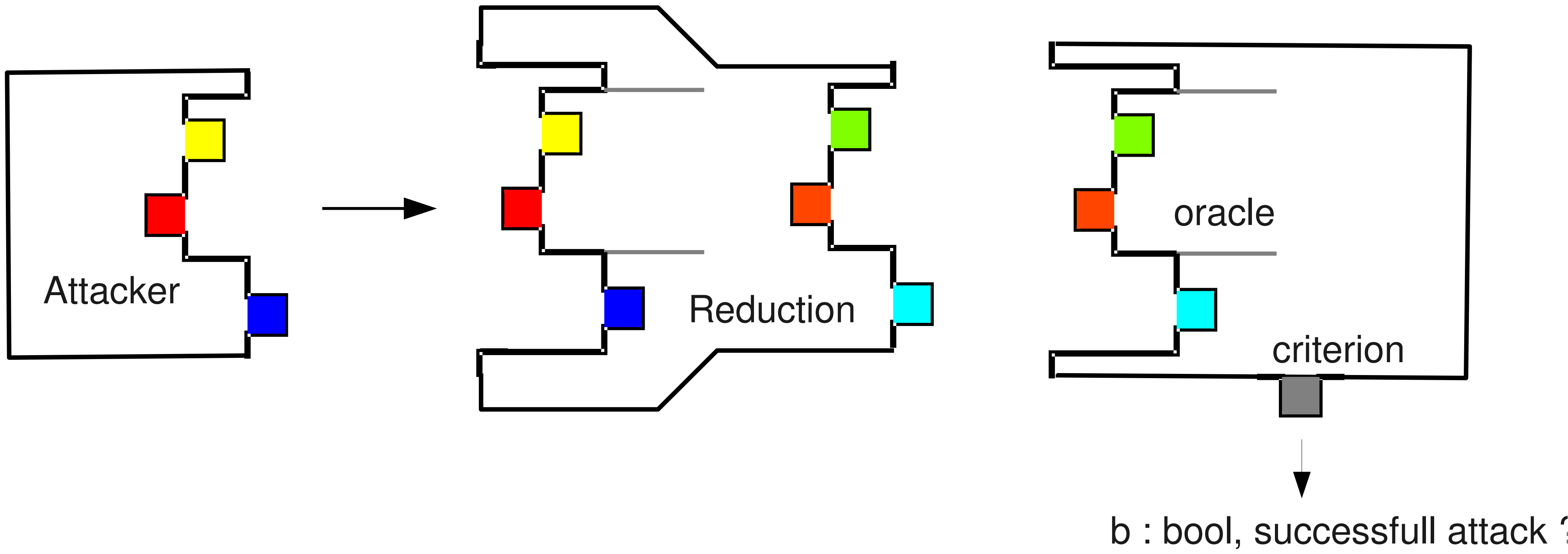
Attacker : oracle → answer (Ordre 2)

Higher order in Cryptography



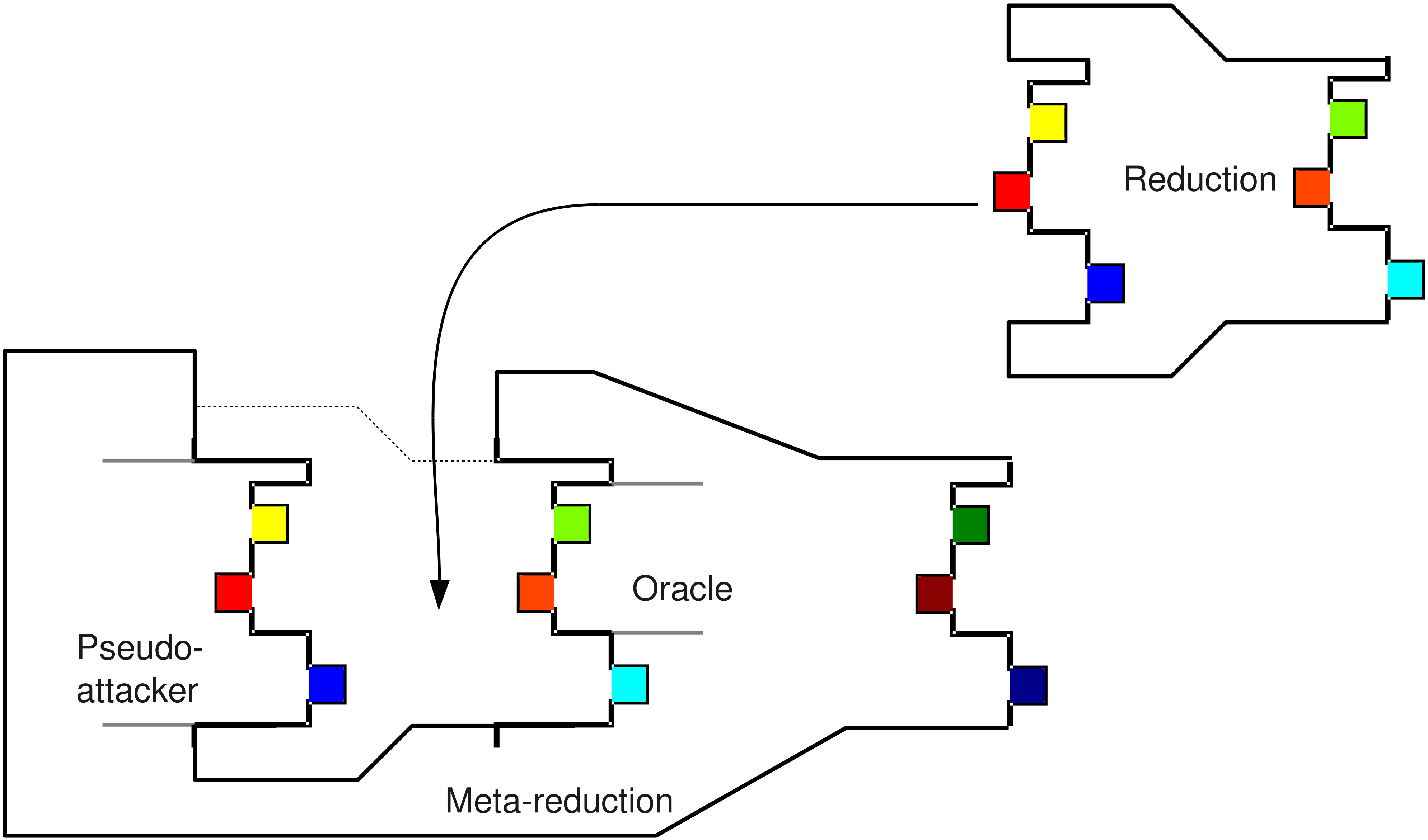
Critère : attacker → bool (Ordre 3)

Higher order in Cryptography



Réduction : attacker → attacker' (Ordre 3)

Higher order in Cryptography



Meta-reduction : reduction → attacker (Ordre 4)

Lambda-Calculus « à la Moggi »

The Syntax :

$t, t_1, t_2 \ldots$	$::=$	x	Variable
	$\mid$	c	Predefined Constant (primitives)
	$\mid$	$\lambda x.t$	Abstraction
	$\mid$	$t_1\, t_2$	Application
	$\mid$	$\text{let } x = t_1 \text{ in } t_2$	Definition
	$\mid$	$\text{let } x \Leftarrow t_1 \text{ in } t_2$	Sequence of computation
	$\mid$	$\text{val}(t)$	Unitary computation

Among predefined constant :

Constructors for integers, lists, trees ...
Primitive induction operators on each types
References (on pure types only)
Randomness generation

NB : no fixpoint operator

Lambda-Calculus « à la Moggi »

Typing rules :

$$\frac{\Gamma_\delta(c) = \sigma \quad \tau \in \mathrm{Inst}(\sigma)}{\Gamma \vdash c : \tau} \qquad \frac{\Gamma(x) = \sigma \quad \tau \in \mathrm{Inst}(\sigma)}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash \lambda x.t : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash t_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash t_1 t_2 : \tau_2}$$

$$\frac{\Gamma \vdash t_1 : \tau_1 \quad \Gamma, x : \mathrm{Gen}_\Gamma(\tau_1) \vdash t_2 : \tau_2}{\Gamma \vdash \mathrm{let}\ x = t_1\ \mathrm{in}\ t_2 : \tau_2}$$

$$\frac{\Gamma \vdash t_1 : T(\tau_1) \quad \Gamma, x : \tau_1 \vdash t_2 : T(\tau_2)}{\Gamma \vdash \mathrm{let}\ x \Leftarrow t_1\ \mathrm{in}\ t_2 : T(\tau_2)} \qquad \frac{\Gamma \vdash t : \tau}{\Gamma \vdash \mathrm{val}(t) : T(\tau)}$$

ref : a → T (Ref a)   (!) deref : Ref a → T a   (:=) assign : Ref a → a → T U
rand_bool : T bool   rand_int : int → T int

Polymorphic types
State monade with references and random tape,
Monadic types
Denotationnal semantic in ***Set***

Implementation examples

3 examples implemented :

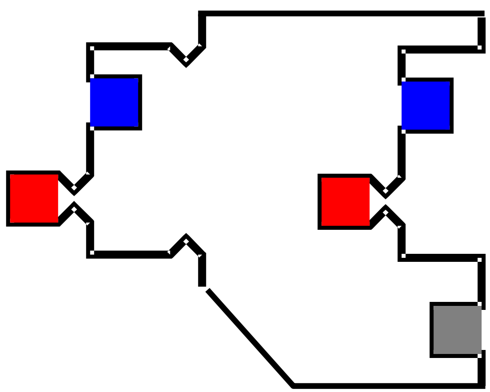
- Hash-Then-Sign construction (as choosed in [Rogaway, 2006])
- Goldreich, Goldwasser & Micali construction (PRG to PRF) [GGM, 1986]
- Meta-reduction of Paillier & Vergnaud [Paillier & Vergnaud, 2005]

Programming style :

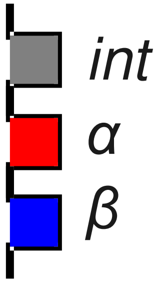
- Re-use of code (modularity)
- Sandboxing references whenever possible
- Think ahead the formal proof

Implementation examples

```
let call_limiter n f =
  let m <= ref n in
  val(fun x ->
       let m1<= !m in
       if (m1 = 0) then exit
       else begin
         m := (m1-1);
         f x
       end
     );;
```

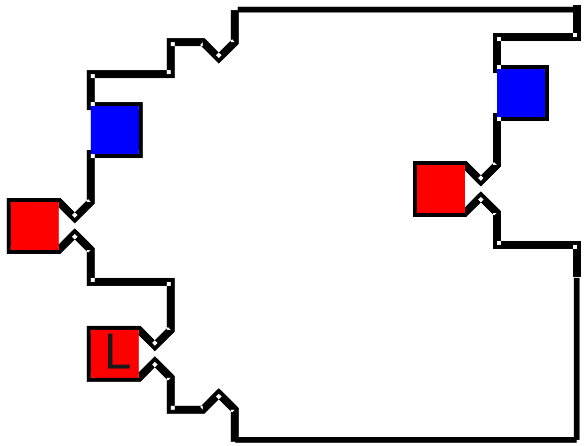


$$\forall\, \alpha\, \beta.\ int \rightarrow (\alpha \rightarrow T\,\beta) \rightarrow T\,(\alpha \rightarrow T\,\beta)$$

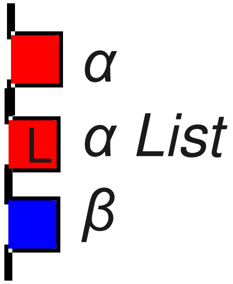


Implementation examples

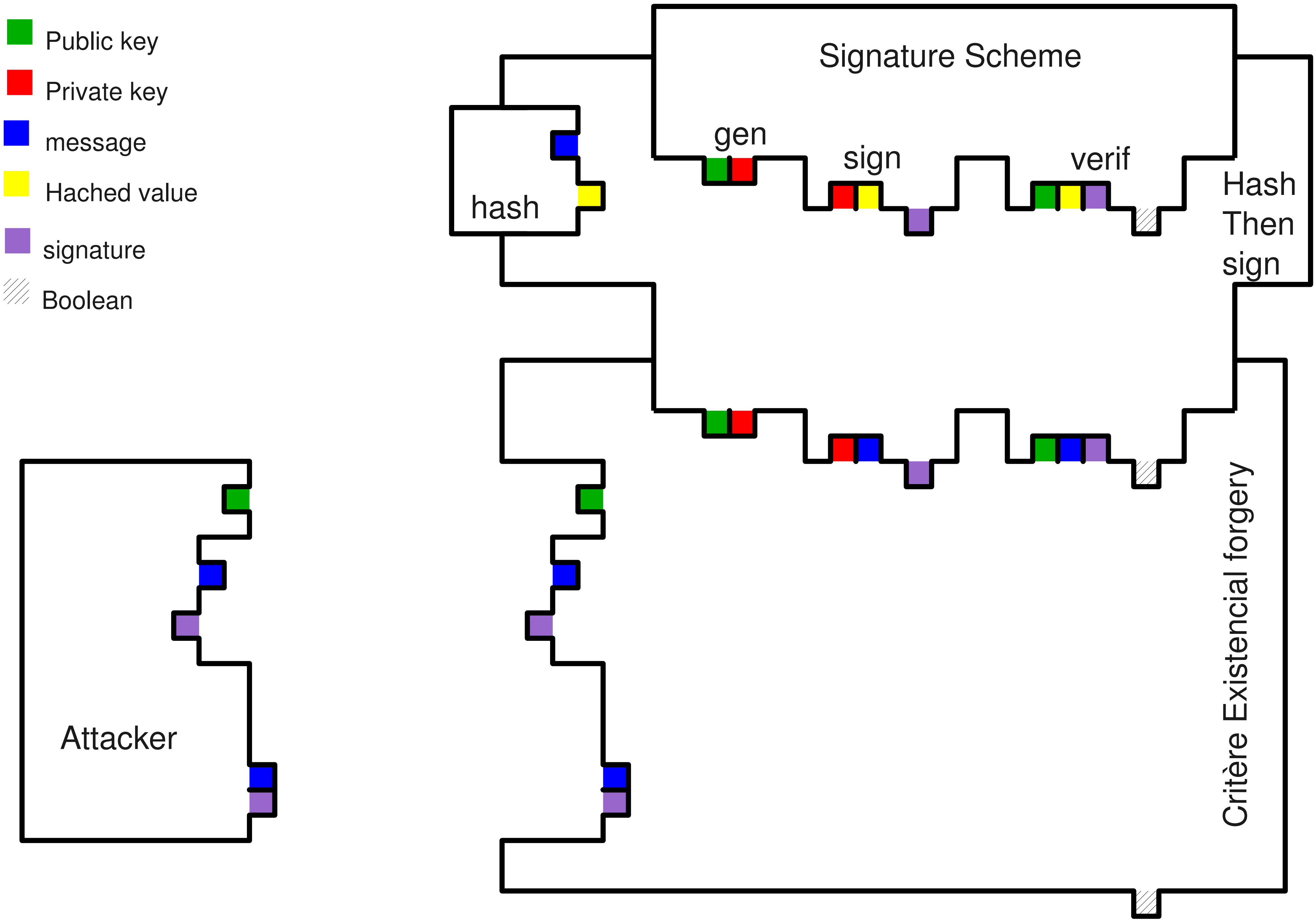
```
let logger f =
  let l <= ref nil in
  Val(
       (fun x -> let ll<= !l in
                 l:=cons x ll;
                 x
       ),
       (!l)
     );;
```



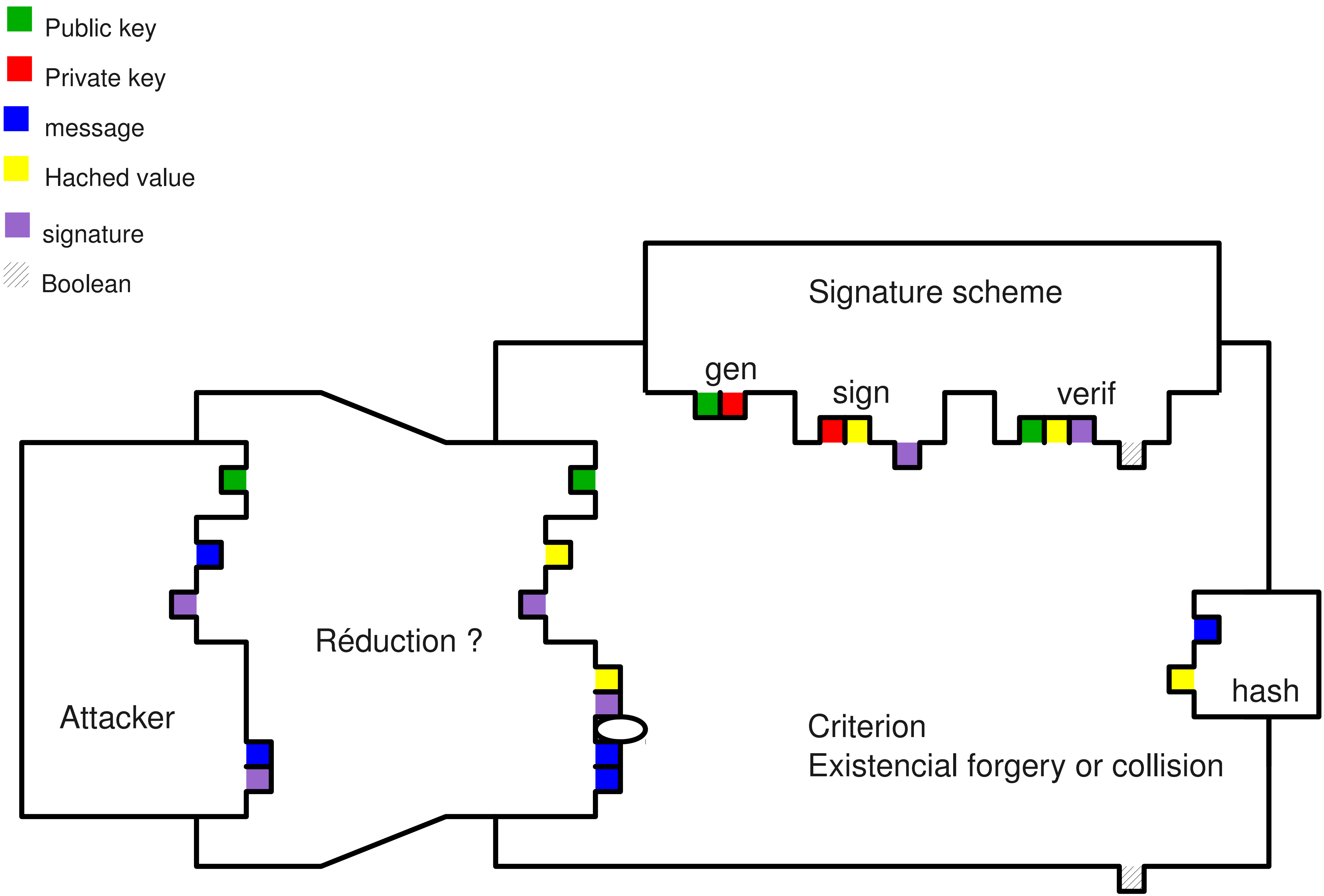
$$\forall\, \alpha\, \beta.\ (\alpha \to T\,\beta) \to T\,((\alpha \to T\,\beta) \times T\,(\alpha\ List))$$



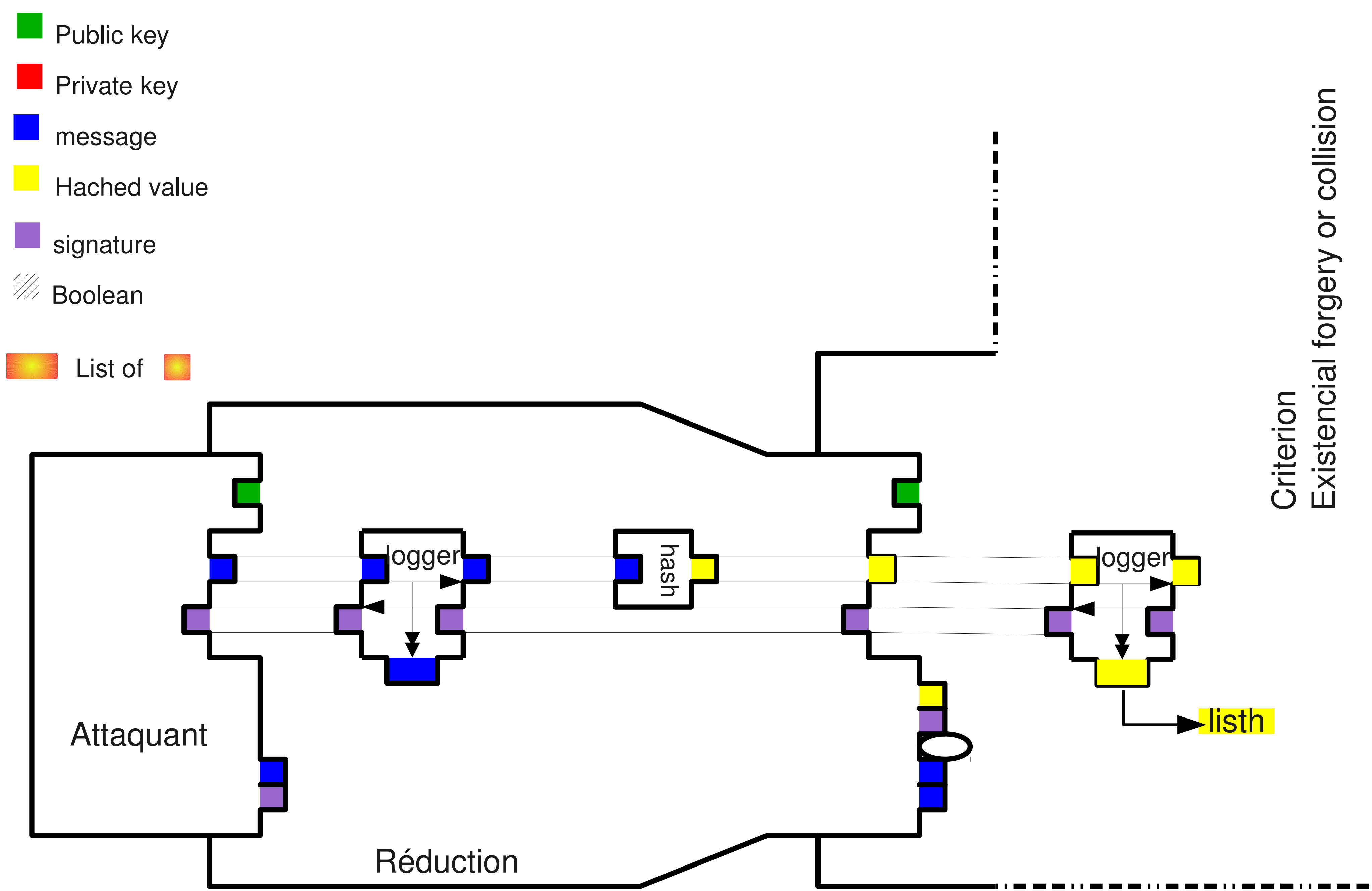
Implementation examples



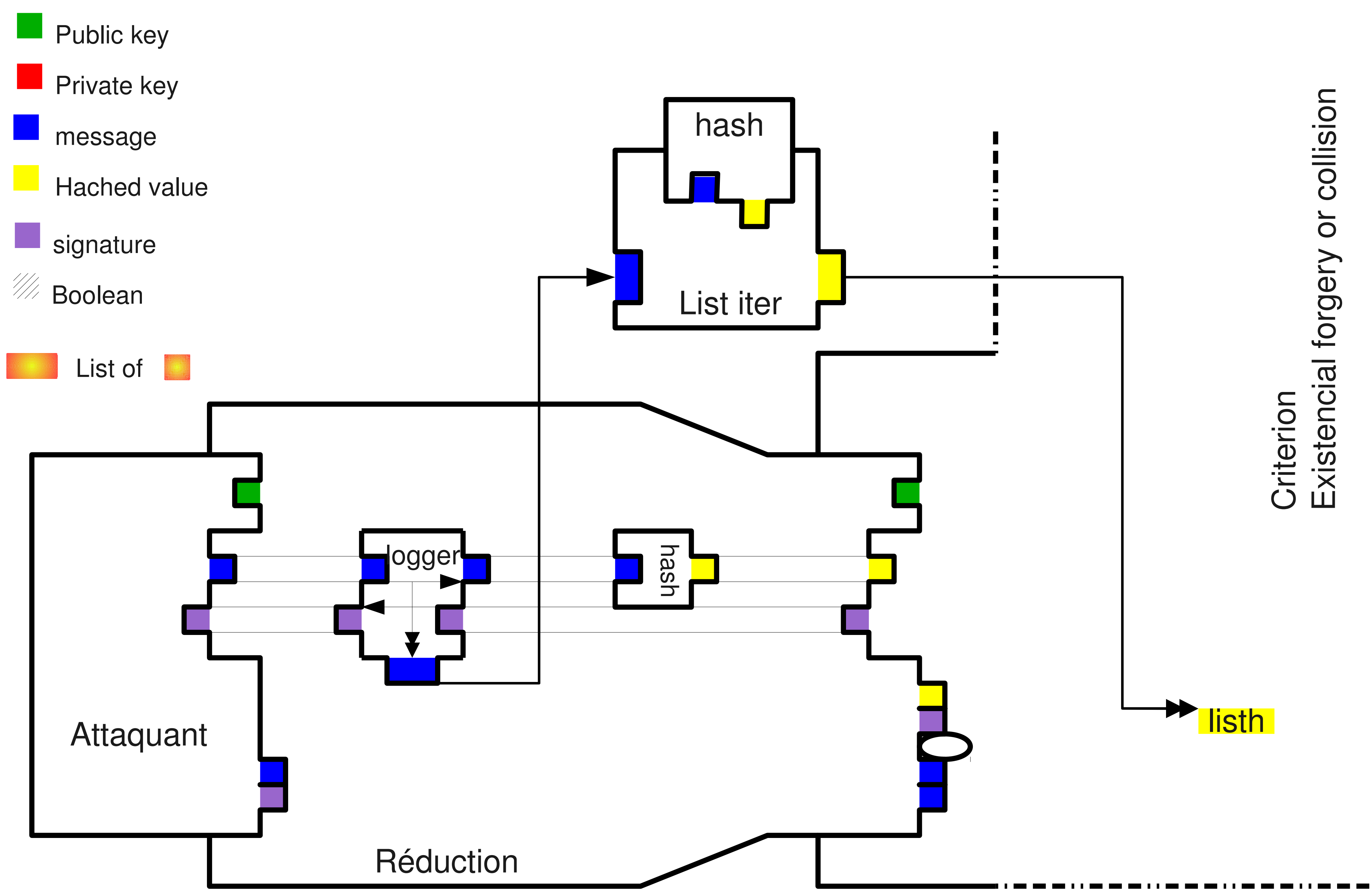
Implementation examples



Implementation examples



Implementation examples

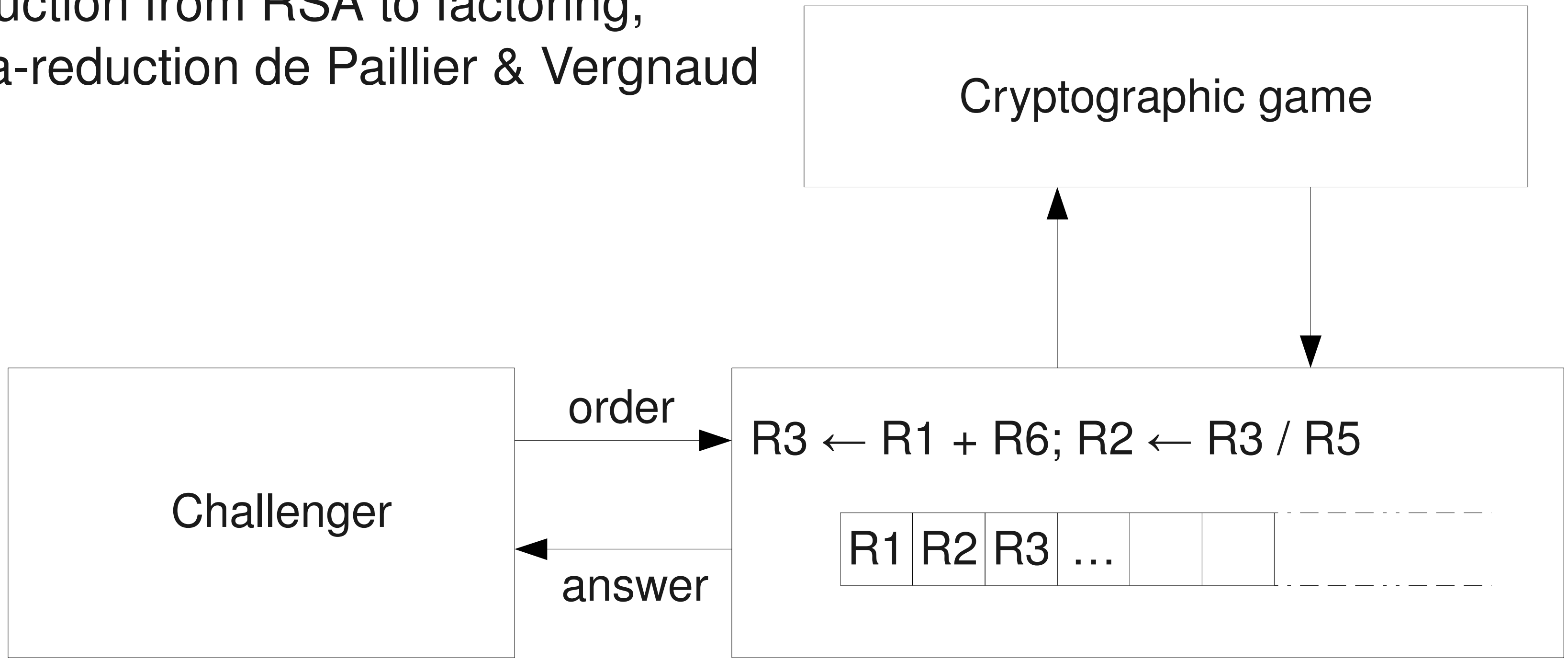


Presentation of algebraic (generic) models

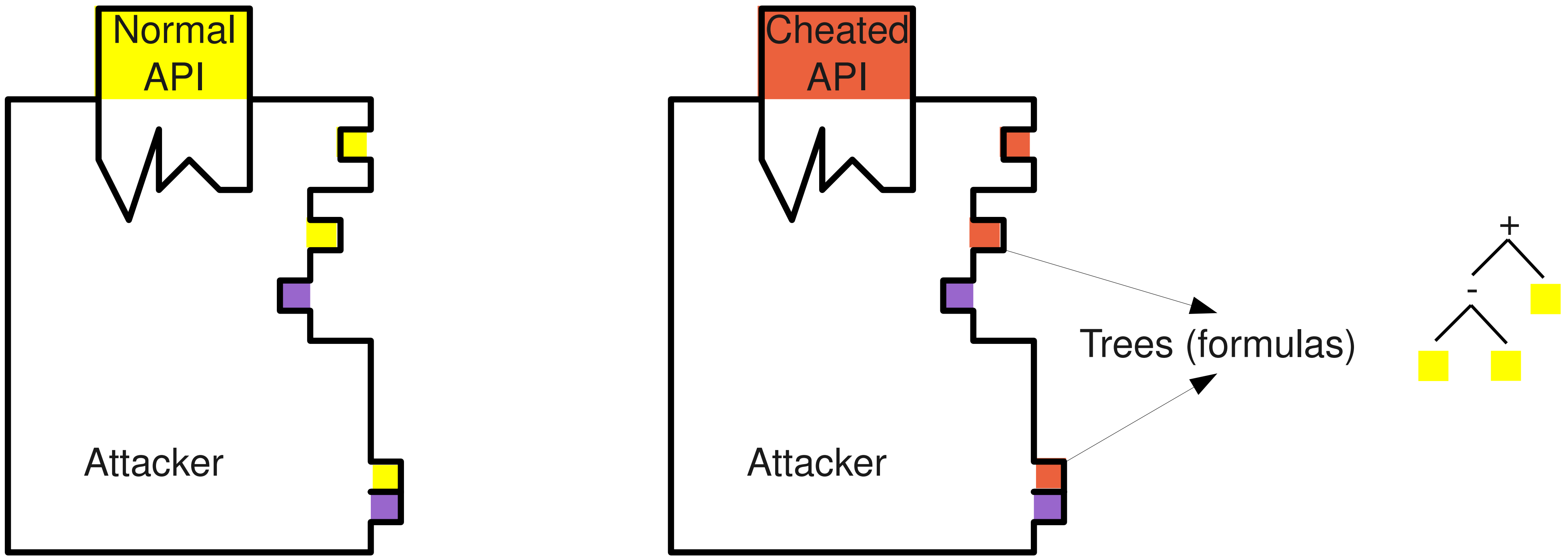
- Restriction of permitted operation (to a certain API)
- Useful to extract information from the attacker (how he build certain objects) and limit its view
- Usually formalised with an intermediate register machine receiving orders

Used in :

Many proofs in the generic group model,
Reduction from RSA to factoring,
Meta-reduction de Paillier & Vergnaud



Taking advantage of polymorphism



Theorem (informal) :

If we replace a normal API by the cheated API, the attacker's behaviour isn't changed much, namely it will output trees instead of normal elements, But such that those trees represent the same elements.
Moreover, thoses trees have for only leaves elements given to the attacker as inputs.

The proof of this theorem used parametricity introduced by [Walder, 1989]

Results

A language with desired property defined

Implementation of interpreter (*Ocaml,* ~ 3000 lines)

- Letting one run and test reductions

Evidence of interest for polymorphic typing

- Re-use of code (re-usability of lemmas on those programs ?)
- Original technique to formalize algebraic models

Other problems

Using pure type (ie. Non-computationnal) to modelize some security definition

Exemple : Key Dependant Message Security, Related-key security

In those models, the attacker choose a function, that will be applied to a secret of the criterion. To modelize properly this, we must not allow the attacker to give a function with side-effect.

Other problems

Extend the language to be able to formalize re-play of an attacker and prove the translated version of the forking lemma

Intuition :

Video game with n levels, with probability one half to complete them,
Failure send back to first level.

Cheat to finish game in polynomial time ?

This idea may be related to :

- Emulation / Virtualization
- Continuation (Lambda-calculus)

Two relaxation of black-boxness possible :

- Reboot, and control source of randomness
- Ability to save/reload the internal state of the attacker

Bibliography

[Bellar & Rogaway, 1994] MWihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. *In Advances in Cryptology – EUROCRYPT 1994*

[Blanchet, 2006] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *In Proc. 2006 IEEE Symposium on Security and Privacy,*

[Backes et al., 2008] Michael Backes, Matthias Berg, and Dominique Unruh. A formal language for cryptographic pseudocode. *In Proc. 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08),*

[Yu Zhang, 2009] Yu Zhang. The computational SLR: a logic for reasoning about computational indistinguishability. *Cryptology ePrint Archive, Report 2008/434 (TLCA '09)*

[Barthe et al., 2009] Gilles Barthe, Benjamin Gr´goire, Romain Janvier, and Santiago Zanella B ´guelin. A framework for language-based cryptographic proofs. *In Proc. 2nd Informal ACM SIGPLAN Workshop on Mechanizing Metatheory, Oct 2007.*

[Rogaway, 2006] Phillip Rogaway. Formalizing human ignorance. *In Progress in Cryptology – VIETCRYPT 2006*

[GGM, 1986] O Goldreich, S Goldwasser, and S Micali. How to construct random functions. *Journal of the ACM, (33) :792–807, 1986.*

[Paillier et Vergnaud, 2005] Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. *In ASIACRYPT, pages 1–20, 2005.*

[Walder, 1989] Philip Wadler. Theorems for free ! *In Proc. 4th Int. Symposium on Functional Programming Languages and Computer Architecture (FPCA'89),*