1. Introduction

Conception of a language for cryptographic reductions

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

Conception of a language for cryptographic reductions
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.
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 assisted / automatisable
But also
Logical and pedagogical interest
1. Introduction

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
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 efficiency (concrete or asymptotic)

Our work focuses on step 1/
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)
1. Introduction

2. The language
   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
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 → answer (Ordre 1)
2. The language

Conception of a language for cryptographic reductions

L. Ducas, Master Thesis
Supervisor: M. Baudet

Higher order in Cryptography

Attacker: oracle $\rightarrow$ answer (Ordre 2)
Higher order in Cryptography

2. The language

Critère : attacker → bool  (Ordre 3)
Réduction : attacker $\rightarrow$ attacker'

(Ordre 3)
2. The language

Conception of a language for cryptographic reductions

L. Ducas, Master Thesis
Supervisor: M. Baudet

Higher order in Cryptography

Meta-reduction: reduction → attacker (Ordre 4)
The Syntax:

\[ t, t_1, t_2 \ldots ::= \]

- \( x \) \hspace{1cm} \text{Variable}
- \( c \) \hspace{1cm} \text{Predefined Constant (primitives)}
- \( \lambda x.t \) \hspace{1cm} \text{Abstraction}
- \( t_1 t_2 \) \hspace{1cm} \text{Application}
- \( \text{let } x = t_1 \text{ in } t_2 \) \hspace{1cm} \text{Definition}
- \( \text{let } x \leftarrow t_1 \text{ in } t_2 \) \hspace{1cm} \text{Sequence of computation}
- \( \text{val}(t) \) \hspace{1cm} \text{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
Typing rules:

\[
\Gamma_\delta(c) = \sigma \quad \tau \in \text{Inst}(\sigma) \quad \Gamma \vdash c : \tau
\]

\[
\Gamma(x) = \sigma \quad \tau \in \text{Inst}(\sigma) \quad \Gamma \vdash x : \tau
\]

\[
\Gamma, x : \tau_1 \vdash t : \tau_2 \quad \Gamma \vdash \lambda x.t : \tau_1 \rightarrow \tau_2
\]

\[
\Gamma \vdash t_1 : \tau_1 \quad \Gamma, x : \text{Gen}_\Gamma(\tau_1) \vdash t_2 : \tau_2
\]

\[
\Gamma \vdash \text{let} \ x = t_1 \ \text{in} \ t_2 : \tau_2
\]

\[
\Gamma \vdash t_1 : T(\tau_1) \quad \Gamma, x : \tau_1 \vdash t_2 : T(\tau_2) \quad \Gamma \vdash t : \tau
\]

\[
\Gamma \vdash \text{let} \ x \leftarrow t_1 \ \text{in} \ t_2 : T(\tau_2)
\]

\[
\Gamma \vdash \text{val}(t) : T(\tau)
\]

ref : a → T (Ref a)  
rand_bool : T bool  

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

Polymorphic types

State monad with references and random tape,

Monadic types

Denotationnal semantic in Set
3 examples implemented:

- Hash-Then-Sign construction (as chosen 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
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
  );;

\( \forall \alpha \beta. \text{int} \to (\alpha \to T \beta) \to T (\alpha \to T \beta) \)
let logger f =
  let l <= ref nil in
Val(
    (fun x -> let ll <= !l in
           l := cons x ll;
          x
     ),
    (!l)
  );;

∀ α β. (α → T β) → T ((α → T β) × T (α List))
2. The language

Implementation examples

- Public key
- Private key
- message
- Hached value
- signature
- Boolean

Signature Scheme

- gen
- sign
- verif

hash

Then

sign

Crittére Existencial forgery

L. Ducas, Master Thesis
Supervisor: M. Baudet
2. The language

Implementation examples

- Signature scheme
- Gen
- Sign
- Verif
- Hash

- Attacker
- Criterion: Existential forgery or collision

- Conception of a language for cryptographic reductions
  Supervisor: M. Baudet
2. The language

Conception of a language for cryptographic reductions
L. Ducas, Master Thesis
Supervisor: M. Baudet
- 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
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, those trees have for only leaves elements given to the attacker as inputs.

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


