CV2EC : Getting the Best of Both Worlds

Bruno Blanchet[1] Pierre Boutry[1] Christian Doczkal[2]
Benjamin Grégoire[1] Pierre-Yves Strub[3]

Inria[1] MPI-SP[2] École Polytechnique[3]

June 16, 2022

Joint Hubert Comon Retirement Workshop - TECAP Workshop

Overview

Why translate from CryptoVerif (CV) to EasyCrypt(EC)?

+ CryptoVerif works well protocol-level verification
+ CryptoVerif is highly automated
− CryptoVerif requires "non-standard" formulations of assumptions
− CryptoVerif cannot do complex reductions (e.g. hybrid proofs)

+ EasyCrypt can express arbitrary reductions
− EasyCrypt proofs are more verbose and less automatic

Overview

Why translate from CryptoVerif (CV) to EasyCrypt(EC)?

+ CryptoVerif works well protocol-level verification
+ CryptoVerif is highly automated
– CryptoVerif requires "non-standard" formulations of assumptions
– CryptoVerif cannot do complex reductions (e.g. hybrid proofs)

+ EasyCrypt can express arbitrary reductions
– EasyCrypt proofs are more verbose and less automatic

Solution: CV2EC

Automatically translate the "non-standard" assumption of CV to EC, and (manually) reduce them to "standard" security assumptions.

CryptoVerif vs EasyCrypt

CryptoVerif	EasyCrypt
Based on π-calculus	Based on pWHILE + Hoare logic
Single-use oracles + replication	Multi-call oracle procedures
all variables are global (arrays indexed by replication indices)	global memory + local variables
Games in "Real/Ideal" style	Can express arbitrary games
Adversary implicit	Adversary explicit

CryptoVerif vs EasyCrypt

CryptoVerif	EasyCrypt
Based on π-calculus	Based on pWHILE + Hoare logic
Single-use oracles + replication	Multi-call oracle procedures
all variables are global (arrays indexed by replication indices)	global memory + local variables
Games in "Real/Ideal" style	Can express arbitrary games
Adversary implicit	Adversary explicit

- Running Example: Real/Ideal formulation of IND-CCA2 assumption (Adversary tries to distinguish *honest* encryption oracle from encryption of *constant message*).

IND-CCA2 Game in EasyCrypt

```
module Game (O : Oracle_i, A : Adversary) = {
  proc main() = {
    O.init();
    r <@ A(O).guess();
    return r;
  }}.
```

```
module type Oracle = {
  proc init() : unit
  proc pk () : pkey
  proc enc (_ : plaintext) : ciphertext
  proc dec (_ : ciphertext)  : plaintext option
}.

module type Adversary (O : Oracle) = {
  proc guess () : bool {O.pk O.enc O.dec}
}.
```

Real Game in EasyCrypt

```
module Real : Oracle_i = {
  var pk : pkey
  var sk : skey

  proc init() : unit = {
    ks <$ dkeyseed;
    pk <- pkgen ks;
    sk <- skgen ks;
  }

  proc pk () = { return pk; }

  proc enc (m : plaintext) : ciphertext = {
    es <$ dencseed;
    return enc(m, pk, es);
  }

  proc dec (c : ciphertext) : plaintext option = {
    return dec(c, sk);
  }
}.
```

Ideal Game in EasyCrypt

```
module Ideal : Oracle_i = {
  ...
  var log : (ciphertext * plaintext) list

  proc init() : unit = {
    ...
    log <- []; }

  proc enc (m : plaintext) : ciphertext = {
    es <$ dencseed;
    c <- enc(m0, pk, es); (* encrypt constant message *)
    log <- (c, m) :: log; (* log provided message *)
    return c; }

  proc dec (c : ciphertext) : plaintext option = {
    m <- assoc log c;
    if (m = None) { m <- dec(c, sk); }
    return m; }
}.
```

IND-CCA2 Assumption in CryptoVerif (Real Game)

```
s <-R keyseed; (
  Opk() := return(pkgen(s))
| foreach i <= N do es <-R enc_seed;
  Oenc(m:plaintext) := return(enc(m, pkgen(s),es))
| foreach i2 <= N2 do
  Odec(c:ciphertext) := return(dec(c, skgen(s))))
```

- sample secret keyseed **`s`**
- provide one copy of the **`Opk()`** oracle
- provide **`N`** copies of the **`Oenc(m)`** oracle (each with some **`enc_seed`**)
- provide **`N2`** copies of the **`Odec(c)`** oracle
- All queries are answered faithfully

IND-CCA2 Assumption in CryptoVerif (Ideal Game)

```
s <-R keyseed; (
  Opk() := return(pkgen(s))
| foreach i <= N do es <-R enc_seed;
  Oenc(m:plaintext) :=
    c_enc:ciphertext <- enc(zero(m), pkgen(s), es);
    return(c_enc)
| foreach i2 <= N2 do
  Odec(c:ciphertext) :=
   find j <= N suchthat
     defined(c_enc[j],m[j]) && c = c_enc[j]
   then return(injbot(m[j]))
   else return(dec(c, skgen(s))))
```

- same replication/oracle signature as real game
- `Oenc(m)` encrypts `zero(m)` (zero message of length `|m|`)
- `Odec(c)` checks whether there is some `j` such that the `j`-th copy of `Oenc` was called and has returned `c`.

Differences

CryptoVerif	EasyCrypt
implicit logging using **`find`**	explicit log using mutable list
sampling of keyseed triggered by adversary (before calling any oracles)	keyseed sampled by game (before calling adversary)
sampling of encseed triggered by adversary before calling **`Oenc`**	encseed sampled by encryption oracle

- Translation yields an EC game encoding CV semantics

Extraction of `Odec()` Oracle

```
(* extra argument i2 corresponding to replication index *)
proc p_Odec(i2 : int, c : ciphertext) = {
  (* check that i2 is fresh and within bounds *)
  if (1 <= i2 <= b_N2 /\ i2 \notin m_Odec) {
    (* ensure s has been sampled *)
    s <@ get_s();
    (* find encryption calls that returned c *)
    j_list <- List.filter
       (fun j => (j \in v_c1 /\ j \in m_Oenc) /\
                  (c = (oget v_c1.[j]))) [1..n];
    if (j_list = []) {
      aout <- (dec c (skgen s));
    } else {
      j <$ drat j_list;
      aout <- (injbot (oget m_Oenc.[j]));
    }
  }
  return aout; }
```

Extraction of Odec() Oracle

```
(* extra argument i2 corresponding to replication index *)
proc p_Odec(i2 : int, c : ciphertext) = {
  (* check that i2 is fresh and within bounds *)
  if (1 <= i2 <= b_N2 /\ i2 \notin m_Odec) {
    (* ensure s has been sampled *)
    s <@ get_s();
    (* find encryption calls that returned c *)
    j_list <- List.filter
       (fun j => (j \in v_c1 /\ j \in m_Oenc) /\
                 (c = (oget v_c1.[j]))) [1..n];
    if (j_list = []) {
      aout <- (dec c (skgen s));
    } else {
      j <$ drat j_list;
      aout <- (injbot (oget m_Oenc.[j]));
    }
  }
  return aout; }
```

This is **not** the IND-CCA2 game in EC!

Differences

CryptoVerif	EasyCrypt
implicit logging using **`find`**	explicit log using mutable list
sampling of keyseed triggered by adversary (before calling any oracles)	keyseed sampled by game (before calling adversary)
sampling of encseed triggered by adversary before calling **`Oenc`**	encseed sampled by encryption oracle

- Translation yields an EC game encoding CV semantics
- Proving the reduction is done manually
  - Eager/Lazy arguments to move sampling
  - replace "find" with explicit logs (for now)
- Pure EC developments: reduce real/ideal EC games to standard assumptions (hybrid arguments, etc.)

Case Studies

- IND-CCA2:
  - ✓ reduction to single challenge query
  - ✓ match EC game with CV output
- Computational Diffie–Hellmann (CDH) for Nominal Groups:
  - ✓ random self-reducibility (from many inputs to one)
  - ✓ match EC game with CV output
- Gap Diffie–Hellmann (GDH) for Nominal Groups:
  - ✓ random self-reducibility (from many inputs to one)
  - ✓ match EC game with CV output
- Outsider-CCA for Authenticated KEMs:
  - ✓ reduction from n users and many encap/decap queries to 2 users and single challenge query.
  - ✓ use explicit logs (not find) in CV games
  - ✓ extend translation to handle CV tables (logs)
  - ✗ match EC game with CV output