CV2EC : Getting the Best of Both Worlds

Bruno Blanchet\textsuperscript{1}   Pierre Boutry\textsuperscript{1}   Christian Doczkal\textsuperscript{2}
Benjamin Grégoire\textsuperscript{1}   Pierre-Yves Strub\textsuperscript{3}

Inria\textsuperscript{1}   MPI-SP\textsuperscript{2}   École Polytechnique\textsuperscript{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

<table>
<thead>
<tr>
<th>CryptoVerif</th>
<th>EasyCrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Based on $\pi$-calculus</td>
<td>Based on pWHILE + Hoare logic</td>
</tr>
<tr>
<td>Single-use oracles + replication</td>
<td>Multi-call oracle procedures</td>
</tr>
<tr>
<td>all variables are global (arrays indexed by replication indices)</td>
<td>global memory + local variables</td>
</tr>
<tr>
<td>Games in “Real/Ideal” style</td>
<td>Can express arbitrary games</td>
</tr>
<tr>
<td>Adversary implicit</td>
<td>Adversary explicit</td>
</tr>
</tbody>
</table>
### CryptoVerif vs EasyCrypt

<table>
<thead>
<tr>
<th>CryptoVerif</th>
<th>EasyCrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Based on $\pi$-calculus</td>
<td>Based on pWHILE + Hoare logic</td>
</tr>
<tr>
<td>Single-use oracles + replication</td>
<td>Multi-call oracle procedures</td>
</tr>
<tr>
<td>all variables are global (arrays indexed by replication indices)</td>
<td>global memory + local variables</td>
</tr>
<tr>
<td>Games in “Real/Ideal” style</td>
<td>Can express arbitrary games</td>
</tr>
<tr>
<td>Adversary implicit</td>
<td>Adversary explicit</td>
</tr>
</tbody>
</table>

- **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

```plaintext
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}
}.
```
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);
  }
}. 
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; }
}.

Bruno Blanchet (Inria)
IND-CCA2 Assumption in CryptoVerif (Real Game)

\[ s \leftarrow_R \text{keyseed}; ( \]
\[ \quad \text{Opk}() := \text{return}(\text{pkgen}(s)) \]
\[ \quad \text{Oenc}(m:\text{plaintext}) := \text{return}(\text{enc}(m, \text{pkgen}(s), \text{es})) \]
\[ \quad \text{Odec}(c:\text{ciphertext}) := \text{return}(\text{dec}(c, \text{skgen}(s))) \]

- sample secret keyseed \( s \)
- provide one copy of the \( \text{Opk}() \) oracle
- provide \( N \) copies of the \( \text{Oenc}(m) \) oracle (each with some \( \text{enc\_seed} \))
- provide \( N_2 \) copies of the \( \text{Odec}(c) \) oracle
- All queries are answered faithfully
IND-CCA2 Assumption in CryptoVerif (Ideal Game)

\[ s \leftarrow_R \text{keyseed}; ( \]
\[ \quad \text{Opk()} := \text{return}(\text{pkgen}(s)) \]
\[ | \quad \text{foreach } i \leq N \text{ do } \text{es} \leftarrow_R \text{enc}_\text{seed}; \]
\[ \quad \text{Oenc}(m:\text{plaintext}) := \]
\[ \quad \quad \text{c_enc:iphertext} \leftarrow \text{enc}(\text{zero}(m), \text{pkgen}(s), \text{es}); \]
\[ \quad \quad \text{return}(\text{c_enc}) \]
\[ | \quad \text{foreach } i2 \leq N2 \text{ do } \]
\[ \quad \text{Odec}(c:\text{ciphertext}) := \]
\[ \quad \quad \text{find } j \leq N \text{ suchthat} \]
\[ \quad \quad \quad \text{defined}(\text{c_enc}[j],m[j]) \land c = \text{c_enc}[j] \]
\[ \quad \quad \text{then return}(\text{injbot}(m[j])) \]
\[ \quad \quad \text{else return}(\text{dec}(c, \text{skgen}(s)))) \]

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

<table>
<thead>
<tr>
<th>CryptoVerif</th>
<th>EasyCrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>implicit logging using <strong>find</strong></td>
<td>explicit log using mutable list</td>
</tr>
<tr>
<td>sampling of keyseed triggered by adversary (before calling any oracles)</td>
<td>keyseed sampled by game (before calling adversary)</td>
</tr>
<tr>
<td>sampling of encseed triggered by adversary before calling <strong>Oenc</strong></td>
<td>encseed sampled by encryption oracle</td>
</tr>
</tbody>
</table>

- Translation yields an EC game encoding CV semantics
Extraction of \texttt{Odec}() Oracle

(* extra argument \texttt{i2} corresponding to replication index *)

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

This is not the IND-CCA2 game in EC!

Bruno Blanchet (Inria)
CV2EC : Getting the Best of Both Worlds
June 16, 2022 10 / 12
Extraction of \texttt{Odec()} Oracle

(* extra argument i2 corresponding to replication index *)

\begin{verbatim}
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 @$\text{get}_s();
    (* find encryption calls that returned c *)
    j_list <- List.filter
      (fun j => (j \in v_c1 \&\& j \in m_Oenc) \&\&
        (c = (oget v_c1.[j]))) [1..n];
    if (j_list = []) {
      aout <- (dec c (skgen s));
    } else {
      j <$> drat j_list;
      aout <- (injbot (oget m_Oenc.[j]));
    }
  }
  return aout; }
\end{verbatim}

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

<table>
<thead>
<tr>
<th>CryptoVerif</th>
<th>EasyCrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>implicit logging using <strong>find</strong></td>
<td>explicit log using mutable list</td>
</tr>
<tr>
<td>sampling of keyseed triggered by adversary (before calling any oracles)</td>
<td>keyseed sampled by game (before calling adversary)</td>
</tr>
<tr>
<td>sampling of encseed triggered by adversary before calling <strong>Oenc</strong></td>
<td>encseed sampled by encryption oracle</td>
</tr>
</tbody>
</table>

- 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