EasyCrypt - Lecture 1 - Introduction

Gilles Barthe

Monday November 24th
Lecture 1: Introduction

- Motivation
- Background
- Approach
- Example
Problems with cryptographic proofs

Proofs are error-prone and flawed

- *In our opinion, many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor.* Bellare and Rogaway, 2004-2006

- *Do we have a problem with cryptographic proofs? Yes, we do [...] We generate more proofs than we carefully verify (and as a consequence some of our published proofs are incorrect).* Halevi, 2005

Gap between algorithms, implementations and machine code

- *Omitting one fine-grained detail from a formal analysis can have a large effect on how that analysis applies in practice.* Degabriele, Paterson, and Watson, 2011

- *Real-world crypto is breakable; is in fact being broken; is one ongoing disaster area in security.* Bernstein, 2013
The code-based game-playing approach

- Provable security
  - mathematical approach to cryptography
    (inspired from complexity theory)
  - reductionist proofs: construction is asymptotically secure if some computational problem is asymptotically hard

- Concrete provable security
  - for every adversary that breaks construction with probability $p$ in time $t$, there exists an adversary that breaks computational assumption with probability $p'$ in time $t'$

- Code-based approach: probabilistic programs everywhere!

- Game-playing approach: sequences of simple steps

- Really? Many steps are not so simple
  - uniformly distributed and independent from adversary’s view
  - compiler optimizations
  - implicit invariants
Computer-aided cryptographic proofs

Halevi (2005) outlines the design of a tool for crypto proofs

- code-based approach
- reasoning principles

Many good ideas, but

- no implementation
- limited generality (example driven)
- weak guarantees (no foundations, large TCB)

Ideally:

- solid foundations
- full and independently verifiable proofs
- practical and accessible to cryptographers
Cryptographic proofs as program verification

Program verification:

- achieves practicality and verifiability
- is supported by solid foundations and tools

But:

- programs are probabilistic
- verification goals over multiple programs
  - verify multiple programs
  - programs are often verified pairwise
    (crypto: reductionist argument, PL: program equivalence)
- program verification is only part of the story:
  - mathematical reasoning: algebra, arithmetic, ...
  - (discrete) probabilities
  - proofs by induction
define a verification paradigm for provable security

provable security

= relational verification of parameterized probabilistic programs

compact proofs that adhere to cryptographic practice

– mundane steps fully automated
– users drive proofs by applying high-level principles
– support for mathematical reasoning

narrow the gap between provable sec. and “real-world”

– reference implementations
– executable code

covering a.o.t. side-channels

leverage existing verification techniques and tools

– SMT solvers, theorem provers, computer algebra systems, etc
– program logics, VC generation, invariant generation, etc
– verified compilers, certifying compilers, etc
EasyCrypt

- Verification framework that integrates functionalities of
  - an interactive proof assistant
  - program verification environment
  and achieves automation via SMT and CAS back-ends
- Focused on cryptography
  - discrete probabilities
  - structured proofs
    - composition/instantiation
    - universal and existential quantification over modules
  - ultimately: complexity analysis
- Generic
  - primitives and protocols
  - game-based and simulation-based
  - random oracle, standard model, etc.
- Nexus of a larger system
Foundations

- Program logics (some overlaps, many interplays)
  - probabilistic relational Hoare logic (pRHL)
    \[ \{P\} \ c_1 \sim c_2 \ \{Q\} \]
  - probabilistic Hoare logic (pHL)
    \[ \{P\} \ c \ \{Q\} \ △ ◁ δ \]

- Program transformations
  - constant folding, loop optimizations, etc

- Soundness w.r.t. set-theoretical semantics

- Program logics are embedded in ambient logic
  - hybrid arguments, generic arguments
Programs

▶ User-extensible expression language
  – sub-distributions, higher-order functions, inductive types, etc.

▶ Imperative language

\[ C ::= \text{skip} \quad \text{skip} \]
\[ \quad \text{V} = \text{E} \quad \text{assignment} \]
\[ \quad \text{V} = \$D \quad \text{random sampling} \]
\[ \quad C; \ C \quad \text{sequence} \]
\[ \quad \text{if } \text{E} \text{ then } C \text{ else } C \quad \text{conditional} \]
\[ \quad \text{while } \text{E} \text{ do } C \quad \text{while loop} \]
\[ \quad \text{V} = \mathcal{F}(\text{E}, \ldots, \text{E}) \quad \text{procedure call} \]

– procedures can be left abstract
Semantics

- The set $\text{distr } A$ of discrete sub-distributions over $A$ is the set of functions $\mu : A \to [0,1]$ such that:
  - $\text{supp}(\mu) = \{ a \in A \mid \mu(a) > 0 \}$ is discrete
  - $\sum_{a \in A} \mu(a) \leq 1$
- The probability of $E \subseteq A$ in $\mu \in \text{distr } A$ is defined as
  $$\sum_{a \in E} \mu(a)$$
- Commands are interpreted as maps from memories to sub-distribution on memories
  $$\llbracket c \rrbracket : \mathcal{M} \to \text{distr } \mathcal{M}$$
- Kleisli operator
  $$\cdot \# : (A \to \text{distr } B) \to (\text{distr } A \to \text{distr } B)$$
pRHL: intuition and preview

- \{P\} \ c_1 \sim c_2 \ \{Q\} \text{ is valid iff for all } m_1, m_2 \in \mathcal{M}, \ P \ m_1 \ m_2 \text{ implies }

\mathcal{L}(Q) \ [c_1]_{m_1} [c_2]_{m_2}

- If \{P\} \ c_1 \sim c_2 \ \{A_{\langle 1 \rangle} \Leftrightarrow B_{\langle 2 \rangle}\} \text{ is valid then for all } m_1, m_2 \in \mathcal{M}, \ P \ m_1 \ m_2 \text{ implies }

\Pr[c_1, m_1 : A] = \Pr[c_2, m_2 : B]

- If \{P\} \ c_1 \sim c_2 \ \{A_{\langle 1 \rangle} \Rightarrow B_{\langle 2 \rangle}\} \text{ is valid then for all } m_1, m_2 \in \mathcal{M}, \ P \ m_1 \ m_2 \text{ implies }

\Pr[c_1, m_1 : A] \leq \Pr[c_2, m_2 : B]

- If \{P\} \ c_1 \sim c_2 \ \{-F_{\langle 2 \rangle} \Rightarrow A_{\langle 1 \rangle} \Rightarrow B_{\langle 2 \rangle}\} \text{ is valid then for all } m_1, m_2 \in \mathcal{M}, \ P \ m_1 \ m_2 \text{ implies }

\Pr[c_1, m_1 : A] - \Pr[c_2, m_2 : B] \leq \Pr[c_2, m_2 : F]
Public-key encryption

Algorithms \((K, E, D)\), s.t.:

- \(E\) takes as inputs a public key and a message, and outputs a ciphertext
- \(D\) takes as inputs a secret key and a ciphertext, and outputs a plaintext
- if \((sk, pk)\) is a valid key pair, \(D_{sk}(E_{pk}(m)) = m\)

```scheme
module type Scheme = {
  proc kg() : pkey * skey
  proc enc(pk:pkey, m:plain) : cipher
  proc dec(sk:skey, c:cipher) : plain
}.
```

(A bit of cheating here: \(D\) may be partial)
module Correct (S:Scheme) = {
    proc main(m:plain) : bool = {
        var pk : pkey;
        var sk : skey;
        var c  : cipher;
        var m' : plain;
        (pk, sk) = S.kg();
        c    = S.enc(pk, m);
        m'   = S.dec(sk, c);
        return (m' = m);
    }
}.

Correctness states that m' = m holds with probability 1.
Adversary

```
module type Adv = {
  proc choose (pk:pkey) : msg * msg
  proc guess (c:cipher) : bool
}.
```

Many reasonings require that adversaries are lossless (i.e. terminate with probability 1)
module CPA (S:Scheme, A:Adversary) = {
    proc main() : bool = {
        var pk, sk, m0, m1, c, b, b';

        (pk, sk) = S.kg();
        (m0, m1) = A.choose(pk);
        b = $\{0,1\}$;
        c = S.enc(pk, b ? m1 : m0);
        b' = A.guess(c);
        return (b' = b);
    }
}

Want that probability of $b' = b$ is close to $\frac{1}{2}$
One-way trapdoor permutations

module type Inverter = {
  proc i(pk : pkey, y : rand) : rand
}.

module OW(F:TP, I :Inverter) ={
  proc main() : bool ={
    var x, x', pk, sk;

    x = $uniform_rand;
    (pk,sk) = KG().;
    x' = I.i(pk,(f pk x));
    return (x' = x);
  }
}.

Assume that probability of $x' = x$ is small
Random oracles (excerpts)

module type Oracle = {
    proc init():unit
    proc o(x:from):to
}. 

theory ROM.

module RO:Oracle = {
    var m : (from, to) map

    proc o(x:from) : to = {
        var y : to;
        y = $uniform_to;
        if (!mem x (dom m)) m.[x] = y;
        return (m.[x]);
    }
}.
Example: Bellare and Rogaway 1993 encryption

- **plain** = \(\{0, 1\}^n\) (bitstrings of length \(n\))
- **rand** = \(\{0, 1\}^k\) (bitstrings of length \(k\))
- **cipher** = \(\{0, 1\}^{n+k}\) (bitstrings of length \(n + k\))

```plaintext
proc enc(pk:pkey, m:plain): cipher = {
    var h, s : plain;
    var r : rand;

    r = \${0, 1}^k;  
    h = H.o(r);  
    s = m \oplus h;  
    return ((f pk r) || s);  
}
```
Security

For every CPA adversary $\mathcal{A}$, there exists an inverter $\mathcal{I}$ so that

$$\Pr[\text{CPA}(\mathcal{BR}, \mathcal{A}) : b' = b] - \frac{1}{2} \leq \Pr[\text{OW}(F, \mathcal{I}) : x' = x]$$

In the next slides, we will adopt some shorthands:

- CPA for $\text{CPA}(\mathcal{BR}, \mathcal{A})$
- OW for $\text{OW}(F, \mathcal{I})$
Proof

Game hopping technique

1. For each hop
   - prove validity of pRHL judgment
   - derive probability claim(s)

2. Obtain security bound by combining claims

3. Check execution time of constructed adversary
Conditional equivalence

\[ \mathcal{E}_{pk}(m) : \\
\begin{align*}
  r &= \{0, 1\}^\ell; \\
  h &= H(r); \\
  s &= h \oplus m; \\
  c &= f_{pk}(r) \parallel s; \\
  \text{return } c;
\end{align*} \]

\[ \mathcal{E}_{pk}(m) : \\
\begin{align*}
  r &= \{0, 1\}^\ell; \\
  h &= \{0, 1\}^k; \\
  s &= h \oplus m; \\
  c &= f_{pk}(r) \parallel s; \\
  \text{return } c;
\end{align*} \]

\[ \{ \top \} \sim \mathsf{CPA} \sim \mathsf{G} \{ (\neg r \in L_H)^2 \} \Rightarrow = \{ b, b' \} \]

Hence:

\[ \Pr[\mathsf{CPA} : b' = b] - \Pr[\mathsf{G} : b' = b] \leq \Pr[\mathsf{G} : r \in L_H^A] \]
Equivalence

\[\mathcal{E}_{pk}(m) : \]
\[r = \{0, 1\}^\ell;\]
\[h = \{0, 1\}^k;\]
\[s = h \oplus m;\]
\[c = f_{pk}(r) \parallel s;\]
return \(c\);

\[\mathcal{E}_{pk}(m) : \]
\[r = \{0, 1\}^\ell;\]
\[s = \{0, 1\}^k;\]
\[h = s \oplus h;\]
\[c = f_{pk}(r) \parallel s;\]
return \(c\);

\[\{\top\} \mathcal{G} \sim \mathcal{G}' \{=\{b,b',L_{AH}\}\}\]

Hence:

\[\Pr[\mathcal{G} : r \in L_{AH}^A] = \Pr[\mathcal{G}' : r \in L_{AH}^A]\]

\[\Pr[\mathcal{G} : b' = b] = \Pr[\mathcal{G}' : b' = b] = \frac{1}{2}\]
Equivalence

\[ \mathcal{E}_{pk}(m) : \]
\[ r = \{0, 1\}^\ell; \]
\[ h = \{0, 1\}^k; \]
\[ s = h \oplus m; \]
\[ c = f_{pk}(r) \parallel s; \]
\[ \text{return } c; \]

\[ \mathcal{E}_{pk}(m) : \]
\[ r = \{0, 1\}^\ell; \]
\[ s = \{0, 1\}^k; \]
\[ h = s \oplus m; \]
\[ c = f_{pk}(r) \parallel s; \]
\[ \text{return } c; \]

\{\top\} \quad \mathcal{G} \sim \mathcal{G}' \quad \{=\{b, b', L_{A_H}\}\} \]

Hence:

\[ \Pr[\text{CPA} : b' = b] - \frac{1}{2} \leq \Pr[\mathcal{G}' : r \in L_{A_H}^A] \]
**Reduction**

<table>
<thead>
<tr>
<th><strong>Game CPA</strong> :</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(sk, pk) = \mathcal{K}();$</td>
</tr>
<tr>
<td>$(m_0, m_1) = A_1(pk);$</td>
</tr>
<tr>
<td>$b = {0, 1};$</td>
</tr>
<tr>
<td>$c^* = E_{pk}(m_b);$</td>
</tr>
<tr>
<td>$b' = A_2(c^*);$</td>
</tr>
<tr>
<td>return $(b' = b)$</td>
</tr>
<tr>
<td><strong>Encryption $E_{pk}(m)$ :</strong></td>
</tr>
<tr>
<td>$r = {0, 1}^\ell;$</td>
</tr>
<tr>
<td>$s = {0, 1}^k;$</td>
</tr>
<tr>
<td>$c = f_{pk}(r) \parallel s;$</td>
</tr>
<tr>
<td>return $c;$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Game OW</strong> :</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(sk, pk) = \mathcal{K}();$</td>
</tr>
<tr>
<td>$y = {0, 1}^\ell;$</td>
</tr>
<tr>
<td>$y' = I(pk, f_{pk}(y));$</td>
</tr>
<tr>
<td>return $(y' = y);$</td>
</tr>
<tr>
<td><strong>Adversary $I(p, x)$ :</strong></td>
</tr>
<tr>
<td>$(m_0, m_1) = A_1(p);$</td>
</tr>
<tr>
<td>$b = {0, 1};$</td>
</tr>
<tr>
<td>$s = {0, 1}^k;$</td>
</tr>
<tr>
<td>$c^* = x \parallel s;$</td>
</tr>
<tr>
<td>$b' = A_2(c^*);$</td>
</tr>
<tr>
<td>$y' = [z \in L_{H}^A \mid f_p(z) = x];$</td>
</tr>
<tr>
<td>return $y'$;</td>
</tr>
</tbody>
</table>

| $\{\top\} \ G' \sim OW \ \{(r \in L_{H}^A)_{\langle 1 \rangle} \Rightarrow (y' = y)_{\langle 2 \rangle}\}$ |

Hence:

$$\Pr[G' : r \in L_{H}^A] \leq \Pr[OW : y' = y]$$
Reduction

**Game CPA:**

\[(sk, pk) = K();\]
\[(m_0, m_1) = A_1(pk);\]
\[b = \{0, 1\};\]
\[c^* = \mathcal{E}_pk(m_b);\]
\[b' = A_2(c^*);\]
\[\text{return } (b' = b)\]

**Encryption \(\mathcal{E}_{pk}(m):\)**

\[r = \{0, 1\}^\ell;\]
\[s = \{0, 1\}^k;\]
\[c = f_{pk}(r) \parallel s;\]
\[\text{return } c;\]

**Game OW:**

\[(sk, pk) = K();\]
\[y = \{0, 1\}^\ell;\]
\[y' = I(pk, f_{pk}(y));\]
\[\text{return } (y' = y);\]

**Adversary \(I(p, x):\)**

\[(m_0, m_1) = A_1(p);\]
\[b = \{0, 1\};\]
\[s = \{0, 1\}^k;\]
\[c^* = x \parallel s;\]
\[b' = A_2(c^*);\]
\[y' = [z \in L_H^A \mid f_p(z) = x];\]
\[\text{return } y';\]

\[
\{ \top \} \xrightarrow{\sim} \text{OW} \{ (r \in L_{H}^A)_{\langle 1 \rangle} \Rightarrow (y' = y)_{\langle 2 \rangle} \}
\]

Hence:

\[
\Pr[\text{CPA} : b' = b] - \frac{1}{2} \leq \Pr[\text{OW} : y' = y]
\]
**Variation on indistinguishability**

For every adversary $A$, there exists an adversary $B$ st

$$\left| \Pr[\text{CPA}(A) : b' = b] - \frac{1}{2} \right| = \Pr[\text{CPA}(B) : b' = b] - \frac{1}{2}$$

By case analysis on $\Pr[\text{CPA}(A) : b' = b] \leq \frac{1}{2}$

- If true, then $B$ returns the result of $A$
- If false, then $B$ returns the negation of the result of $A$