Computer-Assisted Cryptographic Proofs for Low-Level Implementations

François Dupressoir
IMDEA Software Institute, Madrid, Spain

June 3, 2015

The Problems with Cryptographic Proofs

Three sources of problems in practice: the difference between

- algorithm and standard
  - elegant mathematics tend to fall apart when applied to bitstrings;
  - performance, compatibility, efficiency, flexibility, ...
- standard and implementation
  - bugs in crypto, protocol, or more ...
  - side-channels, ill-adapted adversary model, ...
- implementation and executable form
  - compiler bugs, compiler optimizations, ...

The difference between algorithm and standard

Encryption $\mathcal{E}_{\mathrm{OAEP}(pk)}(c)$:
$r \xleftarrow{\$} \{0,1\}^{k_0}$;
$s \leftarrow G(r) \oplus (m \,\|\, 0^{k_1})$;
$t \leftarrow H(s) \oplus r$;
$c \leftarrow \mathsf{f}_{pk}(s \,\|\, t)$;
return c

Decryption $\mathcal{D}_{\mathrm{OAEP}(sk)}(c)$:
$(s,t) \leftarrow \mathsf{f}_{sk}^{-1}(c)$;
$r \leftarrow t \oplus H(s)$;
if $([s \oplus G(r)]_{k_1} = 0^{k_1})$
then $\{m \leftarrow [s \oplus G(r)]^{k}; \}$
else $\{m \leftarrow \perp; \}$
return m

Decryption $\mathcal{D}_{\mathrm{PKCS}(sk)}(lH, c)$:
$b0_{\ 1} \| \ s_{\ dbLen} \| \ t \leftarrow \mathsf{f}_{sk}^{-1}(c)$;
$rM \leftarrow MGF(s, hLen)$;
$r \leftarrow t \oplus rM$;
$dbM \leftarrow MGF(r, dbLen)$;
$DB \leftarrow t \oplus dbM$;
$l_{\ ?} \| \ 0^{?}_{\ ?} \| \ 1_{\ 1} \| \ m \leftarrow DB$;
if $(m \neq \perp \ \wedge b0 = 0 \ \wedge l = lH)$
then $\{m \leftarrow m; \}$
else $\{m \leftarrow \perp; \}$
return m

The difference between algorithm and standard (cont'd)

- Extra checks needed to ensure plaintexts are in the domain of the OWTP . . .
  - . . . in practice, that domain depends on the public key.
- Two hash functions (ROMs) are replaced by a single MGF
  - Not a big deal here (domains are disjoint);
  - May be a big deal in other contexts (PSS, . . .)
- Variable length plaintexts
  - Is the modified scheme supposed to be length-hiding? (Not in this case.)
  - In the model, does the adversary get to choose the length or just to learn it? Does it change anything?
- Additional parameter (label lH)
  - Is this meant to be a labelled encryption scheme?
  - Is the proof even still valid?

Bridging the First Gap

- In general, differences cannot be reconciled: the standard is not just a refinement (or instance) of the algoriithm;
  - At this stage, we don't care much for the change in control-flow.
  - The variable-length plaintext part is more annoying.
- We don't try to bridge that gap: we eliminate it: adapt the security definitions and proofs to talk about the standard.
  - Machine-checked proofs help make sure that the added complexity does not put us in trouble with the proof-checking authorities.
  - Computer-aided proofs + automation help us deal with this additional complexity.

The diff. between standard and implementation

Decryption $\mathcal{D}_{\mathrm{PKCS}(sk)}(lH, c)$:

$b0_{\ 1}||\ s_{\ dbLen}||\ t \leftarrow \mathsf{f}_{sk}^{-1}(c)$;
$rM \leftarrow MGF(s, hLen)$;
$r \leftarrow t \oplus rM$;
$dbM \leftarrow MGF(r, dbLen)$;
$DB \leftarrow t \oplus dbM$;
$l_{\ ?}||\ 0^{?}_{\ ?}||\ 1_{\ 1}||\ m \leftarrow DB$;
if $(m \neq \perp\ \wedge b0 = 0\ \wedge l = lH)$
  then $\{m \leftarrow m;\}$
  else $\{m \leftarrow \perp;\}$
return m

Decryption $\mathcal{D}_{\mathrm{PKCS\text{-}C}(sk)}(res, lH, c)$:

if $(c \in \mathrm{MsgSpace}(sk))$
{ $(b0, s, t) \leftarrow \mathsf{f}_{sk}^{-1}(c)$;
  $h \leftarrow MGF(s, hL)$; $i \leftarrow 0$;
  while $(i < hLen + 1)$
  { $s[i] \leftarrow t[i] \oplus h[i]$; $i \leftarrow i + 1$; }
  $g \leftarrow MGF(r, dbL)$; $i \leftarrow 0$;
  while $(i < dbLen)$
  { $p[i] \leftarrow s[i] \oplus g[i]$; $i \leftarrow i + 1$; }
  $l \leftarrow payload\_length(p)^{*}$;
  if $(b0 = 0^8 \wedge [p]_l^{hLen} =^{*} 0..01 \wedge$
      $[p]_{hLen} =^{*} lH)$
    then
      $\{rc \leftarrow Success;$
      $memcpy(res, 0, p, dbLen - l, l);\}$
    else $\{rc \leftarrow DecryptionError;\}$ }
  else $\{rc \leftarrow CiphertextTooLong;\}$
return rc;

The diff. between standard and implementation (cont'd)

- Replacing "atomic" operations on bitstrings with iterated operations
- Argument- and result-passing by reference
- Those things are easy to deal with...

The diff. between standard and implementation (cont'd)

- Replacing "atomic" operations on bitstrings with iterated operations
- Argument- and result-passing by reference
- Those things are easy to deal with...

What is the adversary model?

The black-box case: solutions

- Code extraction: Blanchet and Cadé'12, Almeida et al.'14
- Model extraction: Aizatulin et al.'12
- Direct proof on F# implementation: Bhargavan et al.'11-??
- By total correctness: *D*'13, Almeida et al.'13, Appel et al.'15

The black-box case: solutions

- Code extraction: Blanchet and Cadé'12, Almeida et al.'14
- Model extraction: Aizatulin et al.'12
- Direct proof on F# implementation: Bhargavan et al.'11-??
- By total correctness: *D*'13, Almeida et al.'13, Appel et al.'15 *

By total correctness

- Increasingly many tools can prove total functional correctness:
  - VCC, written in F#, uses Boogie and Z3;
  - Frama-C, written in OCaml, uses Why3 and any solver it can find;
  - Verified C, written in Coq
- Total correctness at C level + compilation using CompCert gives total correctness at ASM level.
- Honorable mention:
  - Bernstein and Schwabe's tool: dropping generality increases automation and performance.

By total correctness

Switching to whiteboard for a bit

By total correctness

Switching to whiteboard for a bit

What is the adversary model?

Adversary models

- Black-box
  - Trivial view (values)
  - Low-level view (full-control of shared memory locations)
- Side-channels
  - Pure side-channel (timing, power, memory consumption...)
  - In-memory
  - Compromised sibling VM
  - Compromised OS

Additional dimension: competent, stupid, or malicious programmer.

Side-Channels

- In addition to the ouput, the adversary gets some execution-dependent info:
  - execution time,
  - power consumption,
  - EM radiations (visible light, heat, radio),
  - acoustic vibrations...
- The adversary can adaptively choose queries to verify hypotheses on the secrets made from previous observations.

A General Methodology

WIP with Bacelar Almeida, Manuel Barbosa, Gilles Barthe, David Pichardie

Switching to whiteboard for a bit.

A General Methodology

WIP with Bacelar Almeida, Manuel Barbosa, Gilles Barthe, David Pichardie

Switching to whiteboard for a bit.

Will this always work?

Back to OAEP

Decryption $\mathcal{D}_{\text{PKCS-C}(sk)}(res, lH, c)$:

if $(c \in \text{MsgSpace}(sk))$
{ $(b0, s, t) \leftarrow \mathsf{f}_{sk}^{-1}(c)$;
  $h \leftarrow MGF(s, hL)$; $i \leftarrow 0$;
  while $(i < hLen + 1)$
  { $s[i] \leftarrow t[i] \oplus h[i]$; $i \leftarrow i + 1$; }
  $g \leftarrow MGF(r, dbL)$; $i \leftarrow 0$;
  while $(i < dbLen)$
  { $p[i] \leftarrow s[i] \oplus g[i]$; $i \leftarrow i + 1$; }
  $l \leftarrow payload\_length(p)^*$;
  if $(b0 = 0^8 \wedge [p]_l^{hLen} =^* 0..01 \wedge$
      $[p]_{hLen} =^* lH)$
    then
      {$rc \leftarrow Success$;
      $memcpy(res, 0, p, dbLen - l, l)$;}
    else {$rc \leftarrow DecryptionError$;} }
  else {$rc \leftarrow CiphertextTooLong$;}
return rc;

- Consider an adversary that can measure timing;
- (on a cache-free platform);
- (assuming all x86 instructions are constant-time).
- Model: the adversary is given the list of program points traversed while executing the oracle.

Back to OAEP (cont'd)

- Model: the adversary is given the list of program points traversed while executing the oracle.
- On our implementation of OAEP, that leakage is clearly not independent from the secret (final copy-out).
  - We could always make that copy-out oblivious, but you could think about Encrypt-then-MAC, for example.
- How do we deal with this?

Decryption $\mathcal{D}_{\text{PKCS-C}(sk)}(res, c)$:

$\tau \leftarrow \epsilon$;
if $(c \in \text{MsgSpace}(sk))$
{ $\tau \leftarrow L :: \tau$; $(b0, s, t) \leftarrow \mathsf{f}_{sk}^{-1}(c)$;
  $h \leftarrow MGF(s, hL)$; $i \leftarrow 0$;
  while $(i < hLen + 1)$
  { $\tau \leftarrow L :: \tau$; $s[i] \leftarrow t[i] \oplus h[i]$; $i \leftarrow i + 1$; }
  $\tau \leftarrow R :: \tau$; $g \leftarrow MGF(r, dbL)$; $i \leftarrow 0$;
  while $(i < dbLen)$
  { $\tau \leftarrow L :: \tau$; $p[i] \leftarrow s[i] \oplus g[i]$; $i \leftarrow i + 1$; }
  $\tau \leftarrow R :: \tau$; $l \leftarrow payload\_length(p)$;
  if $(b0 = 0^8 \wedge [p]_l^{hLen} = 0..01 \wedge [p]_{hLen} = LHash)$
    {$\tau \leftarrow L :: \tau$; $rc \leftarrow Success$;
    $memcpy(res, 0, p, dbLen - l, l)$;}
  else {$\tau \leftarrow R :: \tau$; $rc \leftarrow DecryptionError$;} }
  else {$\tau \leftarrow R :: \tau$; $rc \leftarrow CiphertextTooLong$;}
return rc, τ;

Transforming the assumptions

- Assumption on leakage: Leakage traces produced when computing $\mathsf{f}_{\mathsf{sk}}^{-1}(c)$ and $\mathsf{f}_{\mathsf{leak}(\mathsf{sk})}^{-1}(c)$ are the same for any c.
- Cryptographic assumption: f^{-1} implementation is secure in the presence of leakage.
- A leak function is "fixed" by the implementation of f^{-1}.
- The result easily applies to various concrete implementations of f^{-1}...
- ...but the final security result may be very weak (or even vacuous).

Proving Security

We do the proof in EasyCrypt.

- First step: abstract away low-level implementation details.
  - Imperative arrays into functional bitstrings,

$$\boxed{\begin{array}{l} i \leftarrow 0; \\ \text{while } (i < hLen + 1) \\ \{\ s[i] \leftarrow t[i] \oplus h[i];\ \ i \leftarrow i + 1;\ \} \end{array}} \rightsquigarrow \boxed{s \leftarrow t \oplus h;}$$

  - $\sim$ 3000 lines of proof - but we've just seen how to automated this.
- Second step: construct the implementation adversary's view from public inputs and outputs at specification level:
  - Construct the black-box view by reading and laying out values into C-like memory;
  - simulate leakage trace using public data (public input and return code, here).
- Third step: specification security proof (simple variant of Fujisaki et al.'s proof)
  - 6 main games, some intermediate games,
  - $\sim$ 3000 lines of proof - This is normal.

CompCert

- CompCert is a certified optimizing C compiler (in Coq).
- It comes with a proof of semantic preservation expressed in terms of (potentially infinite) traces of events.
  - Only terminating programs.
  - Only "safe" programs (no undefined behaviours).
- A trace of events is possible in compiled program iff it is possible in the source program.
- Events can be
  - system calls ("external calls"),
  - I/O from and to the environment, and
  - user-defined events (parameterized by base-typed values).

Compiling PC-secure Programs using CompCert

- User-defined events sufficient to model PC traces.
- They should be sufficient to model more complex leakage models as well.
- Given the chance, compilation could introduce observable differences in the PC trace.
  - A simple static analysis on ASM programs: "There is at least one branching annotation between any two conditional statements."
  - A Coq proof that this is sufficient to efficiently reconstruct PC traces in a 1-1 way.

Some thoughts for future improvements

- Is it always OK to let the leakage depend on output?
  - It depends on the adversary model and the security notion.
  - The definition of "public inputs" may also vary greatly.
- Can we see the low-level view model as a side-channel?
  - It depends on the notion of adversary view you choose.
  - An interative tool to reason about equivalences of (real) programs would be a great thing to have.
- Can we do better than having to write a pure functional specification that bridges the security proof and the functional correctness proof?
  - Not as far as I know.
  - This is particularly painful when the oracles cannot be (easily) expressed as deterministic functions of a random tape.