Certified Machine Code from Provably Secure C-like Code

Towards A Verified Cryptographic Software Toolchain

François Dupressoir

IMDEA Software Institute, Madrid, Spain

Based on joint work with J.C.B. Almeida, M. Barbosa and G. Barthe

Mind the Gap(s)

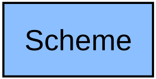
- Cryptographers prove abstract schemes secure.
- Concrete schemes are standardized.
- Implementations are run.

Goal

We aim to bridge these gaps, and bring formal cryptographic guarantees to the level of executable code:

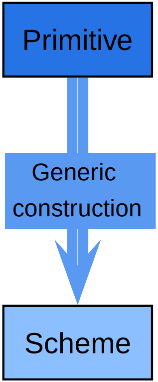
- Perform cryptographic proofs on concrete schemes.
- Certify compilation from schemes to executable code.
- (Along the way, we capture some side-channel leakage.)

Reductionist proof

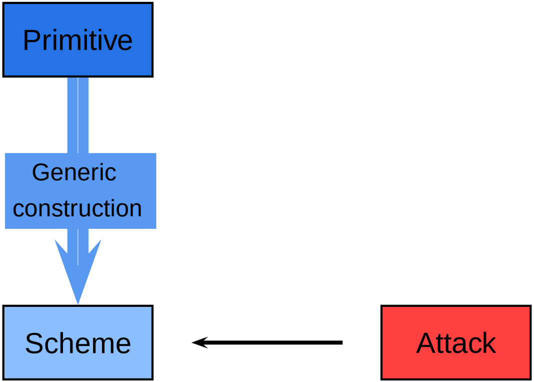


Reductionist proof

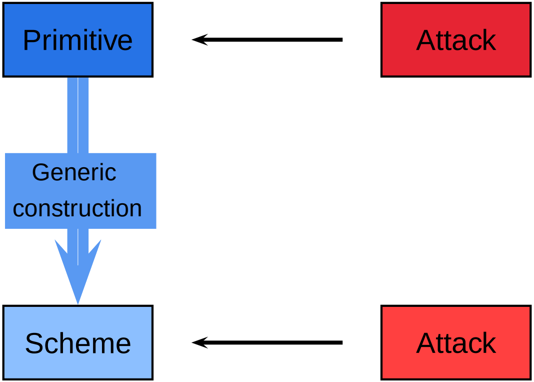
Reductionist proof



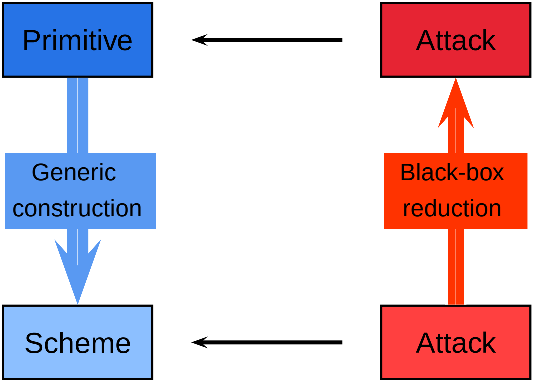
Reductionist proof



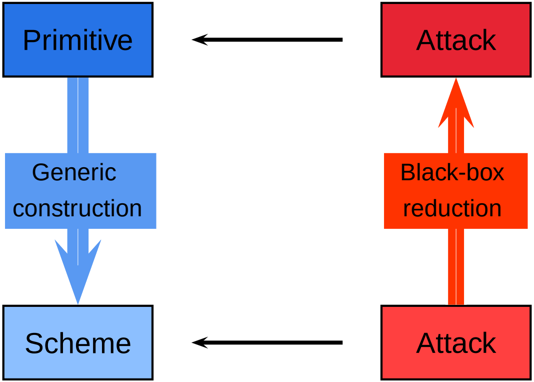
Reductionist proof



Reductionist proof



Reductionist proof



Ideally attacks have similar execution times

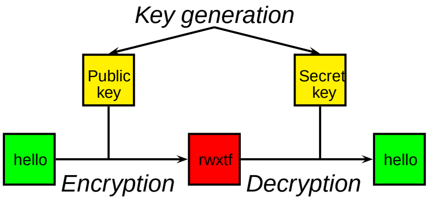
Public-key encryption

Algorithms $(\mathcal{K}, \mathcal{E}_{pk}, \mathcal{D}_{sk})$

- $\mathcal{E}$ probabilistic
- $\mathcal{D}$ deterministic and partial

If (sk, pk) is a valid key pair,

$$\mathcal{D}_{sk}(\mathcal{E}_{pk}(m)) = m$$



Public-key encryption

Indistinguishability against chosen-ciphertext attacks

Game $\mathsf{IND}(\mathcal{A})$
$(sk, pk) \leftarrow \mathcal{K}()$;
$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;
$b \xleftarrow{\$} \{0, 1\}$;
$c^\star \leftarrow \mathcal{E}_{pk}(m_b)$;
$b' \leftarrow \mathcal{A}_2(c^\star)$;
return $(b' = b)$

- $\mathcal{A}_1$ has access to all oracles, and chooses two valid plaintexts *of the same length*.
- $\mathcal{A}_2$ has access to all the oracles (but the decryption oracle fails on $c^\star$) and returns a bit b' representing his guess on the value of b.

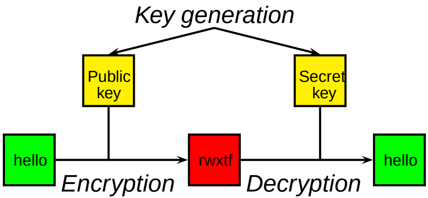
One-way trapdoor permutations

Algorithms $(\mathcal{K}, \mathsf{f}_{pk}, \mathsf{f}_{sk}^{-1})$

- f_{pk} and f_{sk}^{-1} deterministic

If (sk, pk) is a valid key pair,

$$\mathsf{f}_{sk}^{-1}(\mathsf{f}_{pk}(m)) = m$$



One-way trapdoor permutations

set Partial-Domain One-Way

Game sPDOW($\mathcal{I}$)
$(sk, pk) \leftarrow \mathcal{K}()$;
$s \xleftarrow{\$} \{0,1\}^{k_0}$;
$t \xleftarrow{\$} \{0,1\}^{k_1}$;
$x^\star \leftarrow \mathrm{f}_{pk}(s||t)$;
$S \leftarrow \mathcal{I}(pk, x^\star)$;
return $(s \in S)$

- $\mathcal{I}$ is given no oracles but can compute f_{pk} from public data.
- $\mathcal{I}$ returns a list or set of guesses as to the value of s and wins if s is a member.

$$\Pr_{\mathrm{sPDOW}(\mathcal{I})}[s \in S] \quad \text{small}$$

Optimal Asymmetric Encryption Padding

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

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

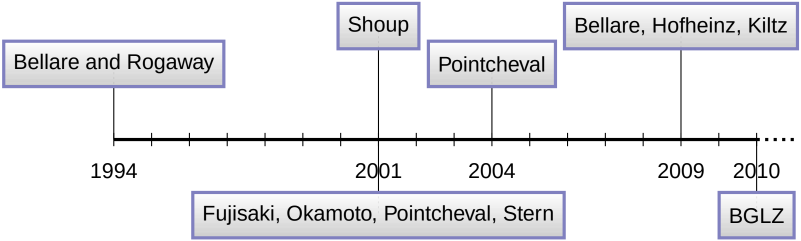
$\oplus$ exclusive or $\|$ concatenation $[\cdot]$ projection 0 zero bitstring

Theorem (Fujisaki et al., 2004)

For every IND-CCA adversary $\mathcal{A}$ against $(\mathcal{K}, \mathcal{E}_{\mathrm{OAEP}}, \mathcal{D}_{\mathrm{OAEP}})$, there exists a set-PDOW adversary $\mathcal{I}$ against $(\mathcal{K}, \mathrm{f}, \mathrm{f}^{-1})$ s.t.

$$\left|\Pr_{\text{IND-CCA}(\mathcal{A})}[b' = b] - \tfrac{1}{2}\right| \leq \Pr_{\text{sPDOW}(\mathcal{I})}[s \in S] + \frac{2q_D q_G + q_D + q_G}{2^{k_0}} - \frac{2q_D}{2^{k_1}}$$

OAEP: Optimal Asymmetric Encryption Padding



1994 Purported proof of chosen-ciphertext security

2001 1994 proof gives weaker security; desired security holds

- for a modified scheme
- under stronger assumptions

2004 Filled gaps in 2001 proof

2009 Security definition needs to be clarified

2010 Fills gaps in 2004 proof

A Low-Level Model...

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 $\mathcal{D}_{\mathrm{PKCS}(sk)}(c)$:
$b0, s, t \leftarrow \mathsf{f}_{sk}^{-1}(c)$;
$rM \leftarrow MGF(s, hL)$;
$r \leftarrow t \oplus rM$;
$dbM \leftarrow MGF(r, dbL)$;
$DB \leftarrow t \oplus dbM$;
$l, m \leftarrow parse(DB)$;
if $(m <> \perp$ &&
   $b0 = 0$ &&
   $l = 0^{hL})$
  then $\{m \leftarrow m; \}$
  else $\{m \leftarrow \perp; \}$
return m

A Lower-Level Model...

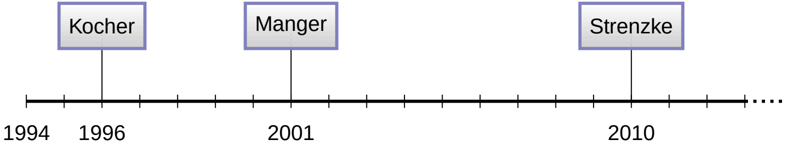
Decryption $\mathcal{D}_{\mathrm{OAEP}(sk)}(c)$:

$(s, t) \leftarrow f_{sk}^{-1}(c)$;
$r \leftarrow t \oplus H(s)$;
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 $\mathcal{D}_{\mathrm{PKCS\text{-}C}(sk)}(res, c)$:

if $(c \in \mathrm{MsgSpace}(sk))$
$\{ (b0, s, t) \leftarrow f_{sk}^{-1}(c)$;
  $h \leftarrow MGF(s, hL)$; $i \leftarrow 0$;
  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 $(b0 = 0^8 \wedge [p]_l^{hLen} = 0..01 \wedge$
    $[p]_{hLen} = LHash)$
    then
      $\{rc \leftarrow Success$;
      $memcpy(res, 0, p, dbLen - l, l); \}$
    else $\{rc \leftarrow DecryptionError; \} \}$
  else $\{rc \leftarrow CiphertextTooLong; \}$
return rc;

A Brief and Incomplete History of Side-Channels



- plaintext is variable-sized: careless parsing leads to padding oracle (Manger, 2001);
- RSA is permutation only on strict subset of $\left[0..2^k\right]$: careless error handling leads to timing attacks;
- PKCS#1 prescribes some error messaging, rarely considered in existing proofs.

...with Leakage

- We consider Program Counter Security.
- The adversary is given the list of program points traversed while executing the oracle.
- Leakage due to the computation of the permutation is kept abstract but given;
- Axioms formalize our leakage assumptions on their implementation.
- Security assumption (sPDOW) is slightly adapted to deal with abstract leakage.

Proving Security

- First step: abstract away low-level implementation details
  - Imperative arrays into functional bitstrings,
  - Separate computation and leakage
  - Loops into abstract operators, easier to reason about.
  - ~3000 lines of proof - This is not nice.
- Then: a variant of Fujisaki et al.'s proof
  - 6 main games, some intermediate games
  - compute cannot handle variable-length bitstrings
  - ~3000 lines of proof - This is normal.

Compilation

- Going from “EasyCrypt C-mode” to C is a syntactic transformation.
  - “C-mode” arrays are base-offset representation and match subset of C arrays (no aliasing or overlap possible, pointer arithmetic only within an array).
  - Some care needed so leakage traces correspond (int as bool, short-circuiting logical connectors).
- Going from C to ASM is more complicated.
- We use CompCert.

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.
  - system calls (“external calls”),
  - I/O from and to the environment, and
  - user-defined events (parameterized by base-typed values).

CompCert and Easycrypt C-mode

- Probabilistic operations pushed into the environment:
  - ideal random sampling of bitstrings,
  - hash function (random oracle),
- Trusted arbitrary precision integer libraries modelled as external calls:
  - some extensions needed to let external calls read and write memory,
  - CompCert and proof extended with “trusted-lib” mechanism,
- User-defined events sufficient to model program counter traces, but may need extensions for other leakage models

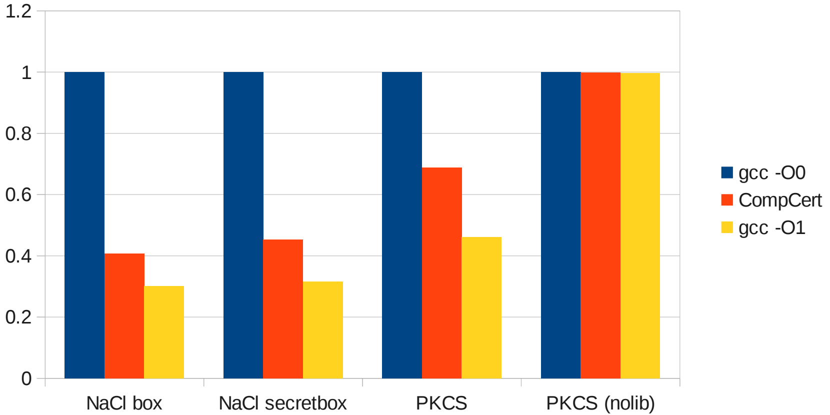
Compiling PC-secure Programs using CompCert

- NaCl functions for sampling and hash functions.
- A simplified variant of LIP for arbitrary precision integers,
  - augmented with PC countermeasures (formally verified),
  - no functional verification.
- Compilation may introduce side-channel (PC) leakage.
  - A simple static analysis on ASM programs,
  - A Coq proof that this is sufficient to guarantee PC-security.

The Check

- There is at least one branching event between any two conditional jumps.
- Guarantees that CompCert traces are in 1-1 relation with PC traces, and that a simulator exists.
- Other leakage models might not enjoy this simplicity.

Performance



- A bit slower than usual CompCert benchmarks,
- Most of the slowdown comes from the trusted library.

Conclusions

Mind the Gap

Still a model.

- Adversary and execution models are still somewhat idealized:
  - Adversary is *not* in the same virtual address space,
  - Initial model is not sufficient to capture cache behaviours, ...
- Consider more active side-channels (fault injection ...)