Verified cryptographic implementations: how far can we go?

Gilles Barthe
IMDEA Software Institute, Madrid, Spain

September 30, 2014

Motivation

- Loss of trust in Internet
  - ☞ Implementation bugs (HeartBleed)
  - ☞ Logical bugs (Triple Handshake)
  - ☞ Backdoors (Dual_EC_DRBG)
  - ☞ Government coercion
- Verification as a (partial) solution: *NIST standard 800-90A is deficient because of a pervasive sloppiness in the use of mathematics. This, in turn, prevents serious mathematical analysis and promotes careless implementation in code. We propose formal verification methods as a remedy*.
  Hales, 2013

Problems with cryptographic proofs

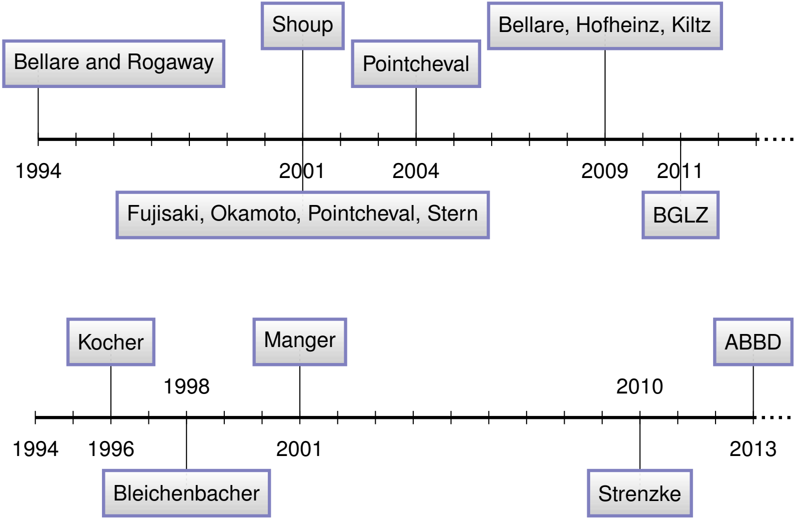
Proofs are error-prone and flawed

- *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, source code 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

OAEP: history



Provable security of OAEP — algorithmic level

Game INDCCA($\mathcal{A}$) :
$(sk, pk) \leftarrow \mathcal{K}(\ )$;
$(m_0, m_1) \leftarrow \mathcal{A}_1^{\mathcal{G},\mathcal{H},\mathcal{D}}(pk)$;
$b \stackrel{\$}{\leftarrow} \{0, 1\}$;
$c^\star \leftarrow \mathcal{E}_{pk}(m_b)$;
$b' \leftarrow \mathcal{A}_2^{\mathcal{G},\mathcal{H},\mathcal{D}}(c^\star)$;
return $(b' = b)$

Provable security of OAEP — algorithmic level

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

Game sPDOW($\mathcal{I}$)
$(sk, pk) \leftarrow \mathcal{K}()$;
$y_0 \xleftarrow{\$} \{0, 1\}^{n_0}$;
$y_1 \xleftarrow{\$} \{0, 1\}^{n_1}$;
$y \leftarrow y_0 \| y_1$;
$x^\star \leftarrow \mathsf{f}_{pk}(y)$;
$Y' \leftarrow \mathcal{I}(x^\star)$;
return $(y_0 \in Y')$

Provable security of OAEP — algorithmic level

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

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

Decryption ...

Game sPDOW($\mathcal{I}$)
$(sk, pk) \leftarrow \mathcal{K}()$;
$y_0 \xleftarrow{\$} \{0,1\}^{n_0}$;
$y_1 \xleftarrow{\$} \{0,1\}^{n_1}$;
$y \leftarrow y_0 \,\|\, y_1$;
$x^\star \leftarrow \mathsf{f}_{pk}(y)$;
$Y' \leftarrow \mathcal{I}(x^\star)$;
return $(y_0 \in Y')$

Provable security of OAEP — algorithmic level

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

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

Decryption ...

Game sPDOW($\mathcal{I}$)
$(sk, pk) \leftarrow \mathcal{K}()$;
$y_0 \xleftarrow{\$} \{0,1\}^{n_0}$;
$y_1 \xleftarrow{\$} \{0,1\}^{n_1}$;
$y \leftarrow y_0 \| y_1$;
$x^\star \leftarrow \mathrm{f}_{pk}(y)$;
$Y' \leftarrow \mathcal{I}(x^\star)$;
return $(y_0 \in Y')$

FOR ALL IND-CCA adversary $\mathcal{A}$ against $(\mathcal{K}, \mathcal{E}_{\mathrm{OAEP}}, \mathcal{D}_{\mathrm{OAEP}})$,
THERE EXISTS a sPDOW adversary $\mathcal{I}$ against $(\mathcal{K}, \mathrm{f}, \mathrm{f}^{-1})$ st

$$\left|\Pr_{\text{IND-CCA}(\mathcal{A})}[b' = b] - \tfrac{1}{2}\right| \leq \Pr_{\text{PDOW}(\mathcal{I})}[y_0 \in Y'] + \frac{3q_D q_G + q_D^2 + 4q_D + q_G}{2^{k_0}} + \frac{2q_D}{2^{k_1}}$$

and

$$t_{\mathcal{I}} \leq t_{\mathcal{A}} + q_D\, q_G\, q_H\, T_{\mathrm{f}}$$

Implementation of OAEP

Decryption $\mathcal{D}_{\text{OAEP}(sk)}(c)$:
$(s, t) \leftarrow f_{sk}^{-1}(c)$;
$r \leftarrow t \oplus H(s)$;
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 $\mathcal{D}_{\text{PKCS-C}(sk)}(res, c)$:
if ($c \in \text{MsgSpace}(sk)$) then
$\{\ (b0, s, t) \leftarrow f_{sk}^{-1}(c)$;
  $h \leftarrow MGF(s, hL)$; $i \leftarrow 0$;
  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;

Computer-aided cryptographic proofs

provable security
=
deductive relational verification of parametrized probabilistic programs

- adhere to cryptographic practice
  - ☞ same proof techniques
  - ☞ same guarantees
  - ☞ same level of abstraction
- leverage existing verification techniques and tools
  - ☞ program logics, VC generation, invariant generation
  - ☞ SMT solvers, theorem provers, proof assistants

EasyCrypt

(B. Grégoire, P.-Y. Strub, F. Dupressoir, B. Schmidt, C. Kunz)

- Initially a weakest precondition calculus for pRHL
- Now a full-fledged proof assistant
  - ☞ proof engine inspired from SSREFLECT
  - ☞ backend to SMT solvers and CAS
  - ☞ embedding rich probabilistic language (w/ modules)
  - ☞ probabilistic Relational Hoare Logic for game hopping
  - ☞ probabilistic Hoare Logic for bounding probabilities
  - ☞ ambient logic
  - ☞ reasoning in the large

A language for cryptographic games

$$
\begin{array}{rcll}
\mathcal{C} & ::= & \text{skip} & \text{skip} \\
& | & \mathcal{V} \leftarrow \mathcal{E} & \text{assignment} \\
& | & \mathcal{V} \xleftarrow{\$} \mathcal{D} & \text{random sampling} \\
& | & \mathcal{C};\ \mathcal{C} & \text{sequence} \\
& | & \text{if } \mathcal{E} \text{ then } \mathcal{C} \text{ else } \mathcal{C} & \text{conditional} \\
& | & \text{while } \mathcal{E} \text{ do } \mathcal{C} & \text{while loop} \\
& | & \mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E}) & \text{procedure call}
\end{array}
$$

- $\mathcal{E}$: (higher-order) expressions
- $\mathcal{D}$: discrete sub-distributions

$\left.\right\}$ user extensible

- $\mathcal{P}$: procedures
  - oracles: concrete procedures
  - adversaries: constrained abstract procedures

Reasoning about programs

- Probabilistic Hoare Logic

$$\models \{P\}c\{Q\} \diamond \delta$$

- Probabilistic Relational Hoare logic

$$\models \{P\} \ c_1 \ \sim \ c_2 \ \{Q\}$$

- Ambient logic

pRHL: a relational Hoare logic for games

- Judgment

$$\vDash \{P\}\ c_1\ \sim\ c_2\ \{Q\}$$

- Validity

$$\forall m_1, m_2.\ (m_1, m_2) \vDash P \implies (\llbracket c_1 \rrbracket\ m_1, \llbracket c_2 \rrbracket\ m_2) \vDash Q^\sharp$$

- Proof rules

$$\frac{\vDash \{P \wedge e\langle 1\rangle\}\ c_1\ \sim\ c\ \{Q\} \qquad \vDash \{P \wedge \neg e\langle 1\rangle\}\ c_2\ \sim\ c\ \{Q\}}{\vDash \{P\}\ \text{if } e \text{ then } c_1 \text{ else } c_2\ \sim\ c\ \{Q\}}$$

$$\frac{P \to e\langle 1\rangle = e'\langle 2\rangle \qquad \vDash \{P \wedge e\langle 1\rangle\}\ c_1\ \sim\ c_1'\ \{Q\} \quad \vDash \{P \wedge \neg e\langle 1\rangle\}\ c_2\ \sim\ c_2'\ \{Q\}}{\vDash \{P\}\ \text{if } e \text{ then } c_1 \text{ else } c_2\ \sim\ \text{if } e' \text{ then } c_1' \text{ else } c_2'\ \{Q\}}$$

+ random samplings, procedures, adversaries…

- Verification condition generator

Deriving probability claims

Assume $\models \{P\}\ c_1 \ \sim\ c_2\ \{Q\}$ and $(m_1, m_2) \models P$

Equivalence

- If $Q \triangleq \bigwedge_{x \in X} x\langle 1\rangle = x\langle 2\rangle$ and $\mathrm{FV}(A) \subseteq X$ then

$$\Pr_{c_1,m_1}[A] = \Pr_{c_2,m_2}[A]$$

- If $Q \triangleq A\langle 1\rangle \Leftrightarrow B\langle 2\rangle$ then

$$\Pr_{c_1,m_1}[A] = \Pr_{c_2,m_2}[B]$$

Conditional equivalence

- If $Q \triangleq \neg F\langle 2\rangle \Rightarrow \bigwedge_{x \in X} x\langle 1\rangle = x\langle 2\rangle$ and $\mathrm{FV}(A) \subseteq X$ then

$$\Pr_{c_1,m_1}[A] - \Pr_{c_2,m_2}[A] \leq \Pr_{c_2,m_2}[F]$$

- If $Q \triangleq \neg F\langle 2\rangle \Rightarrow (A\langle 1\rangle \Leftrightarrow B\langle 2\rangle)$ then

$$\Pr_{c_1,m_1}[A] - \Pr_{c_2,m_2}[B] \leq \Pr_{c_2,m_2}[F]$$

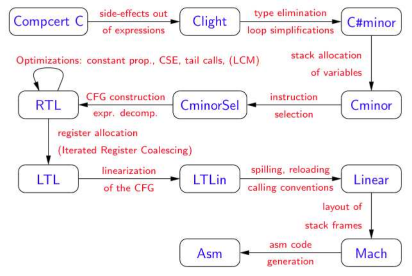
Case studies

- Public-key encryption
- Signatures
- Hash designs
- Block ciphers
- Zero-knowledge protocols
- AKE protocols
- Verifiable computation
- Differential privacy, smart meterting

Provable security of C and executable code

- C-mode using base-offset representation of arrays
  - ☞ no aliasing or overlap possible
  - ☞ pointer arithmetic only within an array
- Reductionist argument for x86 executable code:
  - ☞ FOR ALL adversary that breaks the x86 code,
  - ☞ THERE EXISTS an adversary that breaks the C code
- Use verified compiler to ensure semantic preservation

CompCert (Leroy, 2006)



Security against side-channel attacks

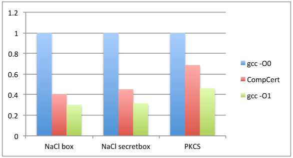
Recipes for security disaster

- Branch on secrets
  - ☞ Lead to timing attacks
  - ☞ PKCS encryption...
- Array accesses with high indices (cache-based attacks)
  - ☞ Lead to cache-based attacks
  - ☞ AES, DES...

- Define static analysis on x86 code
- Extend reductionist argument
  - ☞ FOR ALL adversary that breaks the x86 code,
  - ☞ IF x86 code passes static analysis,
  - ☞ THERE EXISTS an adversary that breaks the C code
- May depend on system-level countermeasures
  - ☞ Use stealth cache for sensitive accesses
  - ☞ Predictive mitigation for timing

Applications to formally verified implementations

- PKCS encryption
  - ☞ INDCCA in the program counter model
  - ☞ Uses constant-time modular exponentiation



- Constant-time cryptography: Salsa, SHA, TEA
- “Almost” constant-time cryptography: AES, DES, RC4
- Vectorized implementations

Challenge

- Highly-optimized implementations are written in assembly
- Cannot use verified compilers
- Alternative: verified decompilers; equivalence checking

Automatic analysis of masked implementations

- Security in t-threshold probing model is non-interference for any t intermediate values
  - Non-interference t intermediate values is a standard program verification model.
  - Easily handled by EasyCrypt.
- Non-interference for any t intermediate values is hard.
  - Size of programs grows with masking order
  - Number of sets to test explodes as masking order grows

Our Solution: Large observation sets

- Given a set of intermediate values known to be safe, efficiently extend it as much a possible.
- Recursively check t non-interference with variables not captured.
- Recursively check t non-interference for sets that straddle both subsets.
- Still exponential, but pretty good in practice.

Improvement: sliding window algorithms

Exploiting the power of refresh gadgets

Intuition: variables are probabilistically independent if they are

- syntactically independent
- dependent, but dependency through many refresh gadgets,

Formally:

- make dependency graph weighted
- define distance between sets of program points (two sets are far away if their distance exceeds the order)
- show that observation sets that can be partitioned into far away sets need not be considered

Key property:

- Inputs and outputs independent, unless intermediate computations is observed

Synthesis of fault attacks

- Increasing need for secure chips
- Must resist physical attacks
- Countermeasures have a cost
- Lack of formal proofs/models
- Sophisticated attacks
  - ☞ physical tampering (laser...)
  - ☞ advanced mathematical algorithms (LLL)

Approach

- Identify post-conditions that could lead to attacks
- Empirically evaluate their complexity
- Use syntax-guided synthesis for finding fault attacks
- Realize attacks

Found several new attacks on RSA and ECDSA signatures

Syntax-guided synthesis

Goal

Given implementation c and fault condition ϕ, find faulted $\widehat{c}$ st

$$\{\top\}\widehat{c}\{\phi\}$$

- Propagate fault condition backwards
- At each step
  - ☞ select real or faulted instruction
  - ☞ compute weakest precondition
  - ☞ perform logical simplifications
- Success if precondition entails computed VC

Issues:

- Loops: use invariant finding techniques
- Search space: use pruning

Example: RSA signatures

function $\textsc{Sign}_{\mathrm{RSA-CRT}}(m)$

  $M \leftarrow \mu(m) \in \mathbb{Z}_N$

  $S'_p \leftarrow \textsc{Exp}_{\textsc{Ladder}}(M \bmod p, d_p, p, q^{-1} \bmod p)$

  $S'_q \leftarrow \textsc{Exp}_{\textsc{Ladder}}(M \bmod q, d_q, q, p^{-1} \bmod q)$

  $S \leftarrow S'_q \cdot p + S'_p \cdot q \bmod N$

  return S

end function

Example: almost full linear combinations

Assume that $N = pq$ such that p, q are prime and $p, q < 2^{n/2}$

Theorem (Informal)

One can efficiently factor N given sufficiently many values S st

$$\exists x, y < 2^{n/2-\varepsilon}.\ S = x \cdot p + y \cdot q$$

Implement attack in SAGE to find minimal number of values ℓ

p, q	512 (bits)				1024 (bits)			
x, y	464	472	480	496	968	976	984	992
ℓ	22	26	33	74	37	44	53	67

Modular exponentiation

```
function ExpLadder(x, e, q, c)
    x̄ ← CIOS(x, R^2 mod q)
    A ← R mod q
    for i = t down to 0 do
        if e_i = 0 then
            x̄ ← CIOS(A, x̄)
            A ← CIOS(A, A)
        else if e_i = 1 then
            A ← CIOS(A, x̄)
            x̄ ← CIOS(x̄, x̄)
        end if
    end for
    A ← CIOS(A, c)
    return A
end function
```

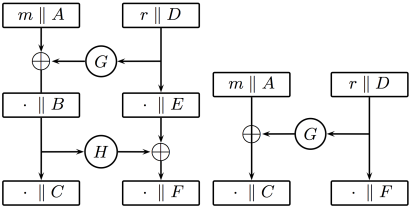
```
function CIOS(x, y)
    a ← 0
    y_0 ← y mod b
    for j = 0 to k − 1 do
        a_0 ← a mod b
        u_j ← (a_0 + x_j · y_0) · q′ mod b
        a ← ⌊(a + x_j · y + u_j · q) / b⌋
    end for
    if a ≥ q then a ← a − q
    end if
    return a
end function
```

- Set $k = 0$ (skip loop)
- Increase value of k and set $q' = 0$ (both $\frac{1}{2}$-exponentiations)
- Double value of k and set $q' = 0$ (one $\frac{1}{2}$-exponentiation)
- Set $q' = 0$ (one $\frac{1}{2}$-exponentiation, Garner recombination)

Synthesis of cryptographic constructions

Do the cryptosystems reflect [...] the situations that are being catered for? Or are they accidents of history and personal background that may be obscuring fruitful developments? [...] We must systematize their design so that a new cryptosystem is a point chosen from a well-mapped space, rather than a laboriously devised construction. (Adapted from Landin, 1966. The next 700 programming languages)

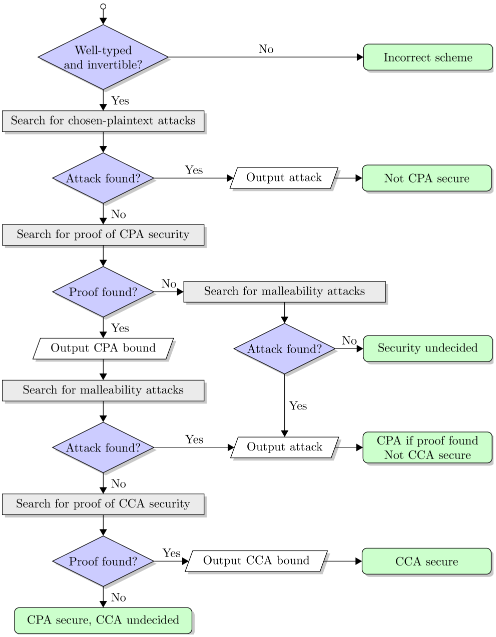
Variants of OAEP



(a) OAEP (b) SAEP

- About 200 variants in the literature
- About $10^6 - 10^8$ candidates schemes of "reasonable" size
- Interactive verification is infeasible (even for 200 schemes)
- Can we automate analysis for finding attacks or proofs?

Approach



Well-typed and invertible?
No
Incorrect scheme
Yes
Search for chosen-plaintext attacks
Attack found?
Yes
Output attack
Not CPA secure
No
Search for proof of CPA security
Proof found?
No
Search for malleability attacks
Attack found?
No
Security undecided
Yes
Yes
Output CPA bound
Search for malleability attacks
Attack found?
Yes
Output attack
CPA if proof found
Not CCA secure
No
Search for proof of CCA security
Proof found?
Yes
Output CCA bound
CCA secure
No
CPA secure, CCA undecided

An algebraic view of padding-based schemes

Encryption algorithms are modelled as algebraic expressions

$$
\begin{array}{rcll}
\mathcal{E} & ::= & m & \text{input message} \\
 & | & 0 & \text{zero bitstring} \\
 & | & \mathcal{R} & \text{uniform random bitstring} \\
 & | & \mathcal{E} \oplus \mathcal{E} & \text{xor} \\
 & | & \mathcal{E} \parallel \mathcal{E} & \text{concatenation} \\
 & | & [\mathcal{E}]_s^s & \text{projection} \\
 & | & H(\mathcal{E}) & \text{hash} \\
 & | & f(\mathcal{E}) & \text{trapdoor permutation}
\end{array}
$$

Decryption algorithms use a mild extension of the language

Attack finding

Apply tools from symbolic cryptography

- Simple filters, eg
  - ☞ is decryption possible without a key? $m \parallel f(r)$
  - ☞ is encryption randomized? $f(m)$
  - ☞ is randomness extractable without a key? $r \parallel f(m \oplus r)$
- Then, static equivalence

$$\frac{e \vdash e_1 \quad e \vdash e_2}{e \vdash e_1 \parallel e_2}[\text{Conc}] \qquad \frac{e \vdash e_1 \qquad e \vdash e_2}{e \vdash e_1 \oplus e_2}[\text{Xor}]$$

$$\frac{e \vdash e}{e \vdash [e]_n^\ell}[\text{Proj}] \qquad \frac{e \vdash e_1 \qquad \vdash e_1 \doteq e_2}{e \vdash e_2}[\text{Conv}]$$

$$\frac{e \vdash e'}{e \vdash H(e')}[\text{H}] \qquad \frac{e \vdash e'}{e \vdash f(e')}[\text{F}] \qquad \boxed{\frac{e \vdash e'}{e \vdash f^{-1}(e')}[\text{Finv}]}$$

Proof finding

Domain-specific computational logic

- Chosen-plaintext security $c :_p \varphi$
- Chosen-ciphertext security $(c, D) :_p \varphi$

Events

- Guess: adversary guesses bit b' correctly
- Ask(e, H): adversary queries hash oracle with e

Few proof principles: for chosen-plaintext security,

- Optimistic sampling: replace $e \oplus r$ by r if r is fresh
- Fundamental Lemma: replace $H(e)$ by fresh r
- Failure event: Ask(e, H) has low prob. if e has high entropy
  - ☞ Symbolic entropy of e: maximal fresh $|\vec{r}|$ st $e \vdash \vec{r}$
- One-wayness: Ask(e, H) has low prob. if reduction exists
  - ☞ Symbolic reduction: do $f(r) \,\|\, m \,\|\, r' \vdash c$ and $e \vdash r$ hold?

Evaluation: chosen-plaintext security

Size	Total	Proof	Attack	Undecided
4	2	1 (50.00%)	1 (50.00%)	0 (0.00%)
5	44	8 (18.18%)	36 (81.82%)	0 (0.00%)
6	335	65 (19.40%)	270 (80.60%)	0 (0.00%)
7	3263	510 (15.63%)	2735 (83.82%)	18 (0.55%)
8	32671	4430 (13.56%)	27894 (85.38%)	347 (1.06%)
9	350111	43556 (12.44%)	301679 (86.17%)	4876 (1.39%)
10	644563	67863 (10.53%)	569314 (88.33%)	7386 (1.15%)
Total	1030989	116433 (11.29%)	901929 (87.48%)	12627 (1.22%)

Evaluation: chosen-ciphertext security

SIZE	PROOF	ATTACK	NR	UNDECIDED
4	0 (0.00%)	2 (100.00%)	0 (0.00%)	0 (0.00%)
5	0 (0.00%)	13 (100.00%)	0 (0.00%)	0 (0.00%)
6	1 (0.98%)	96 (94.12%)	5 (4.90%)	0 (0.00%)
7	45 (5.05%)	739 (82.94%)	45 (5.05%)	62 (6.96%)
8	536 (6.26%)	6531 (76.25%)	306 (3.57%)	1192 (13.92%)
9	7279 (8.16%)	62356 (69.93%)	3035 (3.40%)	16496 (18.50%)
10	20140 (11.29%)	112993 (63.36%)	12794 (7.17%)	32397 (18.17%)
Total	28001 (10.11%)	182730 (65.95%)	16185 (5.84%)	50147 (18.10%)

Minimality in cryptography

- OAEP (1994):

$$f((m\|0) \oplus G(r) \;\|\; r \oplus H((m\|0) \oplus G(r)))$$

- SAEP (2001):

$$f(r \;\|\; (m\|0) \oplus G(r))$$

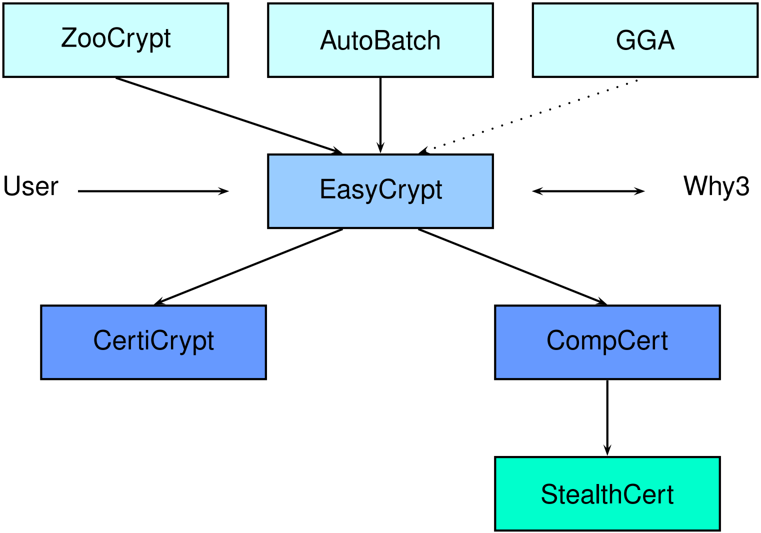
- ZAEP (2012):

$$f(r \,\|\, m \oplus G(r))$$

  ☞ bit-optimal, redundancy-free

  ☞ INDCCA secure for RSA with exponent 2 and 3

EasyCrypt toolchain



Conclusion

- Solid foundation for cryptographic proofs
- Used for emblematic case studies
- Narrowing the gap between proofs and code
- Automated analysis for primitives and assumptions

Further directions

- synthesis and automation (proof theory of cryptography)
- composition and verification of cryptographic systems
- verified implementations (of standards)
- (relational) verification of probabilistic programs: differential privacy, mechanism design, machine learning

`http://www.easycrypt.info`