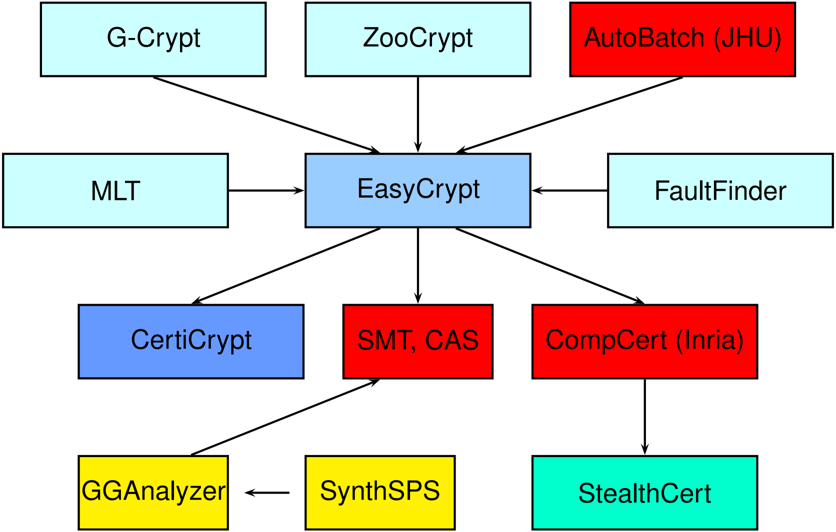
Reconciling provable security and practical cryptography

A programming language perspective

Gilles Barthe
IMDEA Software Institute, Madrid, Spain

May 15, 2015

Tools for high-integrity cryptography



A motivating example: PKCS

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

```
if (c ∈ MsgSpace(sk)) then
{ c0 ← f_sk^{-1}(c);
  (b0, s, t) ← i2bs(c0);
  h ← MGF(s, hLen);  i ← 0;
  while (i < hLen + 1)
  { r[i] ← t[i] ⊕ h[i];  i ← i + 1; }
  g ← MGF(r, dbLen);  i ← 0;
  while (i < dbLen)
  {  p[i] ← s[i] ⊕ g[i];   i ← i + 1; }
  l ← payload_length(p);
  if (b0 = 0^8 ∧ [p]_l^{hLen} = 0..01 ∧ [p]_{hLen} = LHash)
    then
      {rc ← Success;
      memcpy(res, 0, p, dbLen − l, l); }
    else {rc ← DecryptionError; } }
  else {rc ← CiphertextTooLong; }
return rc;
```

f-OAEP

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

Encryption $\mathcal{E}_{\mathrm{OAEP}(pk)}(c)$:

$r \stackrel{\$}{\leftarrow} \{0,1\}^{k_0};$
$s \leftarrow G(r) \oplus (m \| 0^{k_1});$
$t \leftarrow H(s) \oplus r;$
$c \leftarrow \mathsf{f}_{pk}(s \| t);$
return c

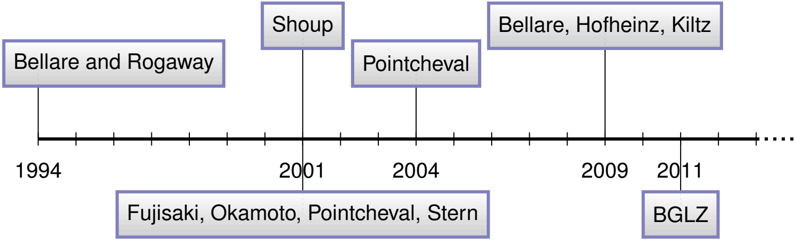
Game INDCCA($\mathcal{A}$) :

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

Game sPDOW($\mathcal{I}$)

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

Provable security of f-OAEP



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

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

and

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

Approach: computer-aided cryptographic proofs

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

(code-based game-playing) provable security
=
deductive relational verification
of parametrized probabilistic programs

EasyCrypt

Next generation program verification environment

- full-fledged proof assistant (inspired from SSREFLECT)
- backend to SMT solvers and CAS
- native embedding of rich probabilistic language
- probabilistic Relational Hoare Logic for game hopping
- probabilistic Hoare Logic for bounding probabilities
- libraries of proof techniques
- module system and theory mechanism
- (soon) automation from symbolic cryptography

Applications

Emblematic examples

- encryption, signatures, hash designs, zero knowledge protocols, garbled circuits, secure function evaluation, verifiable computation
- (computational) differential privacy, mechanism design

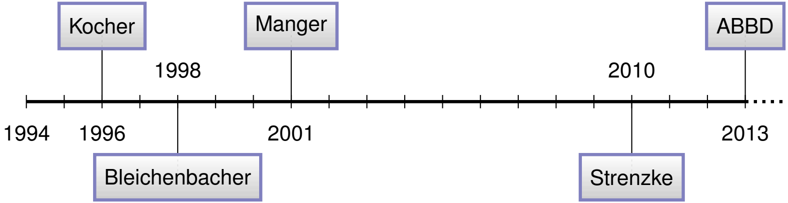
Magic of machine-checked proofs

- synthesis of encryption schemes
- key exchange under weaker assumptions

Ongoing examples

- SHA3
- Voting

Back to PKCS



An isolated case?

- *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

Provable security vs practical cryptography

- Proofs reason about algorithmic descriptions
- Standards constrain implementations
- Attackers target executable code and exploit side-channels

Existing solutions bring limited guarantees

- Leakage-resilient cryptography (mostly theoretical)
- Real-world cryptography (still in the comp. model)
- Constant-time implementations (pragmatic)
- Program transformations (pragmatic)

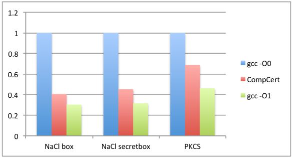
Our approach

- Separation of concerns: establish formal contracts between theoretical and practical cryptographers (and compiler and static analysis writers)
- Strong guarantees on executable code
- Amenable to tool support and machine-checked proofs

Provably secure implementations

Control-flow attacks

- Security proof on C code (EasyCrypt C-mode)
- Reductionist argument (requires semantic preservation)
  - ☞ FOR ALL adversary that breaks the x86 code,
  - ☞ THERE EXISTS an adversary that breaks the C code
- Adding leakage (requires leakage simulation)
  - ☞ Model leakage at C level
  - ☞ Model leakage at assembly level
- Application
  - ☞ PKCS in program counter model



Provably secure implementations

Cache attacks

- Use static analysis on x86 code to prove no leakage
- 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
- Applications to constant-time cryptography
  Salsa, SHA, TEA, AES, DES, RC4...
  (some algorithms need stealth memory)
- Proof relative to an idealized model of virtualization

Provably secure implementations

DPA attacks

- Measuring power consumption allows to retrieve keys
- Masking uses secret sharing to protect against DPA
  - ☞ each input is divided into d shares
  - ☞ computation operates on shares
- Achieves probabilistic non-intereference (PNI) wrt bounded sets of observations: the marginal distribution for any $t \leq d$ observations can be simulated from t shares of each input;
- PNI is easy to check for a fixed set of observations, but hard for all sets of observations is hard. Explosion as masking order d grows:
  - ☞ size of programs increases
  - ☞ number of observation sets explodes

Our Solution

Large observation sets

- given a set of intermediate values known to be safe, efficiently extend it as much a possible
- still exponential, but pretty good in practice

Strong non-interference

- ensures that $t - k$ intermediate values and k outputs can be simulated from $t - k$ shares of each input
- supports compositional principles
- improves efficiency of implementations

Implementation

- automated checker (returns valid or violating tuple)
- certifying compiler
- used to mask AES, Keccak, Simon, Speck at high orders
- generated code is reasonably fast, e.g.
  7-order code is $\sim 100\times$ slower than unmasked code

Benchmarks

Reference	Target	# tuples	Result	Complexity # sets	time (s)
First Order Masking					
CHES10	multiplication	13	secure ✓	7	ε
FSE13	Sbox	63	secure ✓	17	ε
FSE13	full AES	17,206	secure ✓	3,342	128
Second Order Masking					
RSA06	Sbox	1,188,111	secure ✓	4,104	1.649
CHES10	multiplication	435	secure ✓	92	0.001
CHES10	Sbox	7,140	1^{st}-order flaws	866	0.045
CHES10	key schedule	23,041,866	secure ✓	771,263	340,745
FSE13	AES 2 rounds	25,429,146	secure ✓	511,865	1,295
FSE13	AES 4 rounds	109,571,806	secure ✓	2,317,593	40,169
Third Order Masking					
CHES10	multiplication	24,804	secure ✓	1,410	0.033
FSE13	Sbox	4,499,950	secure ✓	33,075	3.894
FSE13	Sbox	4,499,950	secure ✓	39,613	5.036
Fourth Order Masking					
RSA06	Sbox	4,874,429,560	3^{rd}-order flaws	35,895,437	22,119
CHES10	multiplication	2,024,785	secure ✓	33,322	1.138
FSE13	Sbox	2,277,036,685	secure ✓	3,343,587	879
Fifth Order Masking					
CHES10	multiplication	216,071,394	secure ✓	856,147	45

Conclusion

Formal methods provide solid and practical foundations for (reconciling) provable security and practical crypto

Our tools allow to

- formally prove security of cryptographic constructions
- generate correct, secure, and optimized code, which can resist implementation-level adversaries

Further directions

- embedded domain-specific logics
- verified standards and cryptographic systems
- automated discovery of fault attacks
- verification of privacy-preserving computations

`http://www.easycrypt.info`