Computer-aided cryptographic proofs

Gilles Barthe
IMDEA Software Institute, Madrid, Spain

July 18, 2014
Motivation

- Cryptography is a small but important part of security
- Proofs are a small but important part of cryptography
- Hard to get right
- Often iterate over extended period (≥10 years)

- 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
Computer-aided cryptographic proofs

provable security

= 

deductive 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, CAS
  ➤ certified compilers
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 SS\textsc{Reflect}
  - Calls to SMT and CAS
  - Embedding of rich probabilistic language w/ modules
    (neither shallow nor deep)
  - Support for different program logics
  - Reasoning in the large

Applications

- PKCS encryption
- Verification of cryptographic systems
- Key-exchange protocols under weaker assumptions
Reductionist proofs

Assumption

Construction

Definition

Attack

Reduction

Attack
For every INDCPA adversary $\mathcal{A}$, there exists an inverter $\mathcal{I}$ st
\[
\left| \Pr_{\text{INDCPA}(\mathcal{A})} [b' = b] - \frac{1}{2} \right| \leq \Pr_{\text{OW}(\mathcal{I})} [y' = y]
\]
A language for cryptographic games

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

◮ \(E\): (higher-order) expressions
◮ \(D\): discrete sub-distributions
◮ \(P\): procedures

\{ user extensible

. oracles: concrete procedures
. adversaries: constrained abstract procedures
Reasoning about programs

- Probabilistic Hoare Logic
  \[ \vdash \{ P \} c \{ Q \} \diamond \delta \]

- Probabilistic Relational Hoare logic
  \[ \vdash \{ P \} c_1 \sim c_2 \{ Q \} \]

- Ambient logic

**Applications**

Allows deriving judgments of the form

\[ \Pr_{c_1,m_1}[A_1] \diamond \delta \]

or

\[ \Pr_{c_1,m_1}[A_1] \diamond \Pr_{c_2,m_2}[A_2] \]

or

\[ |\Pr_{c_1,m_1}[A_1] - \Pr_{c_2,m_2}[A_2]| \leq \Pr_{c_2,m_2}[F] \]
pRHL: probabilistic relational Hoare logic

- **Judgment**
  \[ \models \{ P \} \ c_1 \sim c_2 \ \{ Q \} \]
  where \( P \) and \( Q \) denote relations on memories

- **Validity**
  \[
  \forall m_1, m_2. \ (m_1, m_2) \models P \iff ([c_1] m_1, [c_2] m_2) \models Q^\#
  \]

- **Definition of \( Q^\# \)** drawn from probabilistic process algebra

**Application**

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

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

\[
\Pr_{c_1,m_1}[A] = \Pr_{c_2,m_2}[A]
\]
Proof rule: assignments and conditionals

Assignments

\[ \frac{\models \{Q\{e\langle1\rangle/x\langle1\rangle\}\{e'\langle2\rangle/x'\langle2\rangle\}\} \quad x \leftarrow e \quad x' \leftarrow e' \quad \{Q\}}{\models \{Q[x\langle1\rangle := e\langle1\rangle]\} \quad x \leftarrow e \quad \text{skip} \quad \{Q\}} \]

Conditionals

\[ \frac{P \Rightarrow e\langle1\rangle = e'\langle2\rangle}{\models \{P \land e\langle1\rangle\} \quad c_1 \sim c'_1 \quad \{Q\}} \quad \models \{P \land \neg e\langle1\rangle\} \quad c_2 \sim c'_2 \quad \{Q\} \]

\[ \frac{\models \{P\} \quad \text{if } e \text{ then } c_1 \text{ else } c_2 \sim \text{if } e' \text{ then } c'_1 \text{ else } c'_2 \quad \{Q\}}{\models \{P \land e\langle1\rangle\} \quad c_1 \sim c \quad \{Q\}} \quad \models \{P \land \neg e\langle1\rangle\} \quad c_2 \sim c \quad \{Q\} \]

\[ \models \{P\} \quad \text{if } e \text{ then } c_1 \text{ else } c_2 \sim c \quad \{Q\} \]
Proof rules: random assignment

Intuition

Let $A$ be a finite set and let $f, g : A \to B$. Define

1. $c = x \leftarrow \mu; y \leftarrow f \ x$
2. $c' = x \leftarrow \mu'; y \leftarrow g \ x$

Then $\llbracket c \rrbracket = \llbracket c' \rrbracket$ (extensionally) iff there exists $h : A \xrightarrow{1-1} A$ st

1. $f = g \circ h$
2. for all $a$, $\mu(a) = \mu'(h(a))$

\[
\begin{align*}
h \text{ is 1-1 and } \forall a, \mu(a) = \mu'(h(a)) & \implies \forall v, Q\{ h \ v / x \langle 1 \rangle \}\{ v / x \langle 2 \rangle \} \ x \leftarrow^\$ \mu \sim x \leftarrow^\$ \mu' \{ Q \}
\end{align*}
\]
Adversaries perform arbitrary sequences of oracle calls (and intermediate computations)

- No functional specification
- Given the same inputs, provide the same outputs
EasyCrypt toolchain

- ZooCrypt
- FaultFinder
- ZKCrypt
- CertiCrypt
- CompCert
- StealthCert

User → EasyCrypt → Why3
ZooCrypt

Automated analysis of padding-based encryption schemes
- Attack finding tool
- Proof search for domain-specific logics
- Interactive tutor
- Generation of EasyCrypt proofs (ongoing)

- Generated $\geq 10^6$ padding-based encryption schemes
- Proved chosen-plaintext security for 11%
- Found attacks for 88%
- About .5% unknowns
- Interactive tutor
Generic Group Analyzer

- Profusion of (non-standard) cryptographic assumptions
  - for efficiency reasons
  - for achieving a construction
- Some assumptions are broken
- Heuristics: prove absence of algebraic attacks
  - Master theorem: security from symbolic condition
  - Use CAS or SMT to discharge symbolic condition

Example: DDH

- Cannot distinguish between \((g^x, g^y, g^{xy})\) and \((g^x, g^y, g^z)\)
- Symbolic condition: \((x, y, xy)\) and \((x, y, z)\) satisfy the same linear equalities
FaultFinder

- Goal: find physical attacks on implementations
- Isolate post-conditions $\phi$ that enable attacks
- Given an implementation $c$, find faulted implementation $\hat{c}$ st
  $$\{\psi\} \hat{c} \{\phi\}$$
- Use SMT-based synthesis
- New attacks for RSA and ECDSA signatures
Conclusion

- Solid foundation for cryptographic proofs
- Formal verification of emblematic case studies

Different styles of proofs
- EasyCrypt: proof objects
- ZooCrypt: proof trees
- GGA: traces
- FaultFinder: proofs for attack finding

Further directions
- Proof Theory of Cryptographic Proofs
- Synthesis of “classical” cryptography

http://www.easycrypt.info