EasyCrypt - Lecture 6
Overview and perspectives

Tuesday November 25th
EasyCrypt - Lecture 6

- Case studies
- Verified implementations
- Automated proofs and synthesis
- Perspectives
Inventaire à la Prevert
Examples with EasyCrypt and friends

► Public-key encryption
  – OAEP, Cramer-Shoup, Boneh-Franklin, Boneh-Boyen...
► Signatures
  – FDH, PSS, CL
► Hash designs
  – Merkle-Damgard, hashing into elliptic curves
► Zero knowledge protocols
  – $\Sigma$-protocols, GSP
► AKE protocols
  – Naxos, HMQV, etc
► Multi-party and verifiable computation
  – Garbled circuits, oblivious transfer, etc
► Differential privacy and mechanism design
  – statistics, vertex cover, MWEM, propose-test-release
► PIR (MIT-LL)
► KEM in TLS handshake (MSR-INRIA-IMDEA)
Verified implementations

**Decryption** \( D_{\text{OAEP}(sk)}(c) \):

\[
(s, t) = f_{sk}^{-1}(c);
\]

\[
r = t \oplus H(s);
\]

if \( ([s \oplus G(r)]_{k_1} \neq 0^{k_1}) \)

then \( \{ m = [s \oplus G(r)]^k; \} \)

else \( \{ m = \bot; \} \)

return \( m \)
Verified implementations

\[ \text{Decryption } D_{PKCS-C}(sk)(c) : \]
\[
\text{if } (c \in \text{MsgSpace}(sk)) \]
\[
\{ (b0, s, t) = f_{sk}^{-1}(c); \]
\[
h = MGF(s, hL); \ i = 0; \]
\[
\text{while } (i < hLen + 1) \]
\[
\{ s[i] = t[i] \oplus h[i]; \ i = i + 1; \} \]
\[
g = MGF(r, dbL); \ i = 0; \]
\[
\text{while } (i < dbLen) \]
\[
\{ p[i] = s[i] \oplus g[i]; \ i = i + 1; \} \]
\[
l = \text{payload\_length}(p); \]
\[
\text{if } (b0 = 0^8 \land [p]_{hLen} = 0..01 \land [p]_{hLen} = LHash) \]
\[
\text{then...} \]
Verified source implementations

Prove “real-world“ security at source level
► Reason about detailed implementations
► Reflect adversarial capabilities in security definitions
► Adapt computational assumptions

Concretely
► “C-mode” arrays are base-offset representation and match subset of C arrays (no aliasing or overlap possible, pointer arithmetic only within an array)
► Explicitly model leakage with instrumented program
Provable security of executable code

- How to carry a proof about C-like code to x86 executable?
- Reduction proof:
  FOR ALL adversary that breaks the executable code,
  THERE EXISTS an adversary that breaks the source code
- Semantic preservation is the crux of proof
  - Prove appropriate semantic preservation in Coq
  - Prove reduction argument on pen-and-paper
- Issues
  - Idealized operations moved to the environment
    - random sampling of bitstrings
    - hash function (random oracle)
  - “trusted-lib” mechanism for arithmetic libraries
CompCert (Leroy, 2006)

- Optimizing C compiler implemented in Coq
- Formal proof of semantic preservation
Security in the PC model

- Branching on secrets is a well-known recipe for disaster
- Program counter model proves absence of high branches
- Oracles are modified to return the sequence of program points traversed during execution (use CompCert annotations)
- Leakage due to the computation of RSA is axiomatized and also given to the adversary in the INDCCA game
- Assumption of RSA security is also adapted to leakage
- Security proof using standard tools
- Check on x86 that compiler does not introduce branching
- “trusted-lib” must also verify leakage assumptions
Constant-time cryptography

- Secret-dependent memory accesses are another well-known recipe for disaster
- Define a constant-time info. flow analysis for x86
- Prove
  - FOR ALL adversary that breaks the x86 code,
  - IF x86 code passes static analysis,
  - AND x86 code and C code semantically equivalent,
  - THERE EXISTS an adversary that breaks the C code
- Benefits:
  - applies to handwritten code
- Soundness proof based on
  - idealized model of virtualization
  - including stealth memory
Masked implementations

- Protect implementations against higher-order DPA
- Security in the \( t \)-probing model: for every \( t \) intermediate program points \( pc_1 \ldots pc_t \), the distribution
  \[
  (x_{pc_1}, \ldots, x_{pc_t})
  \]
  is independent of secrets
- Probabilistic non-interference! Can be checked using pRHL
- Compositional definition is simulation-based: for every \( t \) intermediate program points \( pc_1 \ldots pc_t \), the distribution
  \[
  (x_{pc_1}, \ldots, x_{pc_t})
  \]
  can be computed from inputs \((v_1, \ldots, v_\ell)\) with \( \ell \leq t \)
- Compiler (in progress)
Automated proofs and synthesis

- Impedance mismatch
  - different abstraction levels
  - big de Bruijn factor

Use high-level principles, invariant inference, etc.

- The next 700 cryptosystems

  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)
Approach

- **Attack finding**
  - symbolic methods: deducibility, static inequivalence

- **Proof finding**
  - high-level proof principles as proof rules
  - proof search (possibly guided)
  - algebraic methods
  - symbolic methods
    - symbolic entropy
    - symbolic reduction

- **Synthesis**
  - Fix bugdet (number of operations)
  - Attack then prove
  - Proof generation

- **Tutor**

- **ZAEP** $f(r \parallel m \oplus G(r))$
Batch implementations

- Perform multiple verifications at once (signatures, ZK proofs)
- Small Exponent Test:
  - let $p$ be prime such that $2^\ell \leq p$
  - let $a_i, b_i \in \mathbb{F}_p$ s.t. $a_i \neq b_i$ for some $i$
  - let $x_i$ is sampled uniformly over $(0 \ldots 2^{\ell-1})$

Then

$$\Pr \left[ \sum_{i<n}[x_i]a_i = \sum_{i<n}[x_i]b_i \right] \leq 2^{-\ell}$$

- Equational reasoning
Examples using AutoBatch/EasyCrypt

**Waters 09 signature scheme**

\[ e([b]g_1, \sum_{i=1}^{\eta} [s_i \cdot \delta_i] \sigma_{i,1}) + e([b \cdot a_1]g_1, \sum_{i=1}^{\eta} [s_i,1 \cdot \delta_i] \sigma_{i,2}) \\
+ e([a_1]g_1, \sum_{i=1}^{\eta} [s_i,1 \cdot \delta_i] \sigma_{i,3}) + e([b \cdot a_2]g_1, \sum_{i=1}^{\eta} [s_i,2 \cdot \delta_i] \sigma_{i,4}) \\
+ e(g_1^{a_2}, \sum_{i=1}^{\eta} [s_i,2 \cdot \delta_i] \sigma_{i,5}) \\
= e(\sum_{i=1}^{\eta} [\delta_i \cdot s_i,1] \sigma_{i,6}, \tau_1) \\
+ e(\sum_{i=1}^{\eta} [\delta_i \cdot s_i,2] \sigma_{i,6}, \tau_2) + e(\sum_{i=1}^{\eta} [\delta_i \cdot s_i,1] \sigma_{i,7}, [b] \tau_1) \\
+ e(\sum_{i=1}^{\eta} [\delta_i \cdot s_i,2] \sigma_{i,7}, [b] \tau_2) + e(\sum_{i=1}^{\eta} [(\delta_i \cdot -t_1 + \theta_i \cdot \delta_i \cdot \text{tag}_{i,c} \cdot t_i)] \sigma_{i,7}, \omega) \\
+ e(\sum_{i=1}^{\eta} [\theta_i \cdot \delta_i \cdot M_i \cdot t_i] \sigma_{i,7}, u) + e(\sum_{i=1}^{\eta} [\theta_i \cdot \delta_i \cdot t_i] \sigma_{i,7}, h) \\
+ e(g_1, \sum_{i=1}^{\eta} [-t_i \cdot \theta_i \cdot \delta_i] \sigma_{i,k}) + [\sum_{i=1}^{\eta} s_i,2 \cdot \delta_i] A \]

**Groth-Sahai proofs**

\[ \sum_{j=1}^{\mu} e([r_{1,1}] \sum_{i=1}^{\eta} [\alpha_{i,j}] c_{1,i} + [r_{2,1}] (A_j + \sum_{i=1}^{\eta} [\alpha_{i,j}] c_{2,i}), d_{1,j}) \\
+ \sum_{j=1}^{\mu} e([r_{1,2}] \sum_{i=1}^{\eta} [\alpha_{i,j}] c_{1,i} + [r_{2,2}] (A_j + \sum_{i=1}^{\eta} [\alpha_{i,j}] c_{2,i}), d_{2,j}) \\
+ \sum_{j=1}^{\eta} e([r_{1,2}] c_{1,i} + [r_{2,2}] c_{2,i}, B_i) \\
= e([r_{1,1}] u_1,1 + [r_{2,1}] u_1,2, \pi_{1,1}) + e([r_{1,1}] u_2,1 + [r_{2,1}] u_2,2, \pi_{2,1}) \\
+ e([r_{1,1}] \theta_1,1 + [r_{2,1}] \theta_1,2, \nu_{1,1}) + e([r_{1,1}] \theta_2,1 + [r_{2,1}] \theta_2,2, \nu_{2,1}) \\
+ e([r_{1,2}] u_1,1 + [r_{2,2}] u_1,2, \pi_{1,2}) + e([r_{1,2}] u_2,1 + [r_{2,2}] u_2,2, \pi_{2,2}) \\
+ e([r_{1,2}] \theta_1,1 + [r_{2,2}] \theta_1,2, \nu_{1,2}) + e([r_{1,2}] \theta_2,1 + [r_{2,2}] \theta_2,2, \nu_{2,2}) + [r_{2,2}] T \]
Automated analysis of Diffie-Hellman assumptions

- **Proliferation of assumptions**
  - Discrete logarithms
  - Pairings
  - Multilinear maps

- **Continued use of non-standard assumptions is likely**
  - for efficiency reasons
  - often the sole way of achieving a construction
  - no clear “standard” assumption in advanced settings

- **Issues**
  - assumptions might be broken (mLRSW, CDDH...)
  - relationship between assumptions unclear

- **Generic group model:**
  - idealized model to analyze assumptions
  - security is reduced to algebraic problem

- **Generic Group Tool:** use SMT and CAS
Perspectives

EasyCrypt

- High-level proof principles
- Libraries
- Complexity
- Verified source implementations
- Verified standards

Back and front-ends

- Synthesis and automated algebraic reductions
- Certified static analyses
- Automated transformations and verified assumptions
- High-speed cryptography
  (hand-written and/or vectorized implementations)
Conclusion

- Crypto is a great application domain for PL and PV
- PL and PV provide solid foundations for crypto proofs
- Computer-aided proofs address issues in crypto
- Take up
  - Halevi 2005: “Wouldn’t you like to be cited by half of the papers appearing in CRYPTO 2010? Here is your chance…”
  - Still unlikely to hold next school in Stade de France
- But many exciting opportunities
- Check our web page www.easycrypt.info
- Contact us easycrypt-club@lists.gforge.inria.fr