Computer-aided cryptographic proofs

Gilles Barthe
IMDEA Software Institute, Madrid, Spain

March 24, 2014
The two views of cryptography

Computational cryptography
- Strong ties with complexity theory
- Feasible adversary breaks scheme with small probability
- Design of secure and efficient primitives and protocols
- Complex and manual proofs
The two views of cryptography

**Computational cryptography**
- Strong ties with complexity theory
- Feasible adversary breaks scheme with small probability
- Design of secure and efficient primitives and protocols
- Complex and manual proofs

**Symbolic cryptography** *(Dolev and Yao, 1983)*
- Assume perfect cryptography
- Adversary cannot win
- Discovery of logical bugs in protocols
- Strong ties with verification
- Automated proofs
Reconciling the two views
(Abadi and Rogaway, 2000)

Computational soundness
Security in symbolic model implies computational security

- ... under non-standard assumptions on primitives
- Symbolic tools deliver asymptotic guarantees
- ... but no concrete guarantees
- Applicable to many settings
- ... but some impossibility results
Computational proof systems

Prove security
- directly in the computational model
- under standard assumptions

- Indistinguishability logics
- CryptoVerif (game-playing approach for applied $\pi$-calculus)
- EasyCrypt (code-based game-playing approach)
- Domain-specific logics
Why bother?

- 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
## Optimal Asymmetric Encryption Padding

### Encryption $\mathcal{E}_{\text{OAEP}}(pk)(m)$:
- $r \xleftarrow{\$} \{0, 1\}^{k_0}$;
- $s \leftarrow G(r) \oplus (m \parallel 0^{k_1})$;
- $t \leftarrow H(s) \oplus r$;
- Return $f_{pk}(s \parallel t)$

### Decryption $\mathcal{D}_{\text{OAEP}}(sk)(c)$:
- $(s, t) \leftarrow f^{-1}_{sk}(c)$;
- $r \leftarrow t \oplus H(s)$;
- If $([s \oplus G(r)]_{k_1} = 0^{k_1})$
  - Then $\{m \leftarrow [s \oplus G(r)]^{k_1}\}$
  - Else $\{m \leftarrow \bot\}$
- Return $m$

For all IND-CCA adversary $\mathcal{A}$ against $(\mathcal{K}, \mathcal{E}_{\text{OAEP}}, \mathcal{D}_{\text{OAEP}})$, there exists a sPDOW adversary $\mathcal{I}$ against $(\mathcal{K}, f, f^{-1})$ such that

$$\left| \Pr_{\text{IND-CCA}(\mathcal{A})}[b' = b] - \frac{1}{2} \right| \leq$$

$$\Pr_{\text{PDOW}(\mathcal{I})}[y \in Y'] + \frac{3q_D q_G + q_D^2 + 4 q_D + q_G}{2^{k_0}} + \frac{2q_D}{2^{k_1}}$$

And

$$t_{\mathcal{I}} \leq t_{\mathcal{A}} + q_D q_G q_H T_f$$
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
2011  Fills gaps in 2004 proof
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
  - Symbolic cryptography
A language for cryptographic games

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

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

\[\vdash \{P\} \ c_1 \sim c_2 \ {Q} \quad \vdash \{Q\} \ c'_1 \sim c'_2 \ {R}\]

\[\vdash \{P\} \ c_1; c'_1 \sim c_2; c'_2 \ {R}\]

\[\vdash \{P \land e\langle 1\rangle\} \ c_1 \sim c \ {Q} \quad \vdash \{P \land \neg e\langle 1\rangle\} \ c_2 \sim c \ {Q}\]

\[\vdash \{P\} \text{ if } e \text{ then } c_1 \text{ else } c_2 \sim c \ {Q}\]

\[P \rightarrow e\langle 1\rangle = e'\langle 2\rangle\]

\[\vdash \{P \land e\langle 1\rangle\} \ c_1 \sim c'_1 \ {Q} \quad \vdash \{P \land \neg e\langle 1\rangle\} \ c_2 \sim c'_2 \ {Q}\]

\[\vdash \{P\} \text{ if } e \text{ then } c_1 \text{ else } c_2 \sim \text{ if } e' \text{ then } c'_1 \text{ else } c'_2 \ {Q}\]

\[f \text{ is 1-1}\]

\[\vdash \forall v, Q[x\langle 1\rangle := f \ v, x\langle 2\rangle := v] \ x \xleftarrow{\$} A \sim x \xleftarrow{\$} A \ {Q}\]
Applications

Allows deriving judgments of the form

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

or

\[ |\Pr_{c_1,m_1}[A_1] - \Pr_{c_2,m_2}[A_2]| \leq \Pr_{c_2,m_2}[F] \]

Benefits

- Application of rules directed by syntax (mostly)
- Can be automated
- Generate verification conditions
- VCs can be discharged by SMT solvers

Building on 40 years of research in program verification!
Example: Bellare and Rogaway 1993 encryption

**Game** IND-CPA($\mathcal{A}$) :

1. $(sk, pk) \leftarrow \mathcal{K}();$
2. $(m_0, m_1) \leftarrow \mathcal{A}_1(pk);$
3. $b \leftarrow \{0, 1\};$
4. $c^* \leftarrow \mathcal{E}_{pk}(m_b);$
5. $b' \leftarrow \mathcal{A}_2(c^*);$
6. return $(b' = b)$

**Encryption** $\mathcal{E}_{pk}(m):$

$r \leftarrow \{0, 1\}^\ell;$
$s \leftarrow H(r) \oplus m;$
$y \leftarrow f_{pk}(r) \parallel s;$
return $y$

For every IND-CPA adversary $\mathcal{A}$, there exists an inverter $\mathcal{I}$ st

$$\text{Pr}_{\text{IND-CPA}(\mathcal{A})}[b' = b] - \frac{1}{2} \leq \text{Pr}_{\text{OW}(\mathcal{I})}[y' = y]$$
Proof

Game hopping technique

1. For each hop
   - prove validity of pRHL judgment
   - derive probability claims
   - (possibly) resolve some probability expressions using pHL

2. Obtain security bound by combining claims

3. Check execution time of constructed adversary
Conditional equivalence

\[ E_{pk}(m) : \]
\[ r \xrightarrow{\$} \{0, 1\}^\ell; \]
\[ h \leftarrow H(r); \]
\[ s \leftarrow h \oplus m; \]
\[ c \leftarrow f_{pk}(r) \parallel s; \]
\[ \text{return } c \]

\[ \mathcal{E}_{pk}(m) : \]
\[ r \xrightarrow{\$} \{0, 1\}^\ell; \]
\[ h \xrightarrow{\$} \{0, 1\}^k; \]
\[ s \leftarrow h \oplus m; \]
\[ c \leftarrow f_{pk}(r) \parallel s; \]
\[ \text{return } c \]

\[ \models \{T\} \text{ IND-CPA} \sim \mathbb{G} \left\{ (\neg r \in L_H^A) \langle 2 \rangle \rightarrow = b, b' \right\} \]

\[ \Pr_{\text{IND-CPA}}[b' = b] - \Pr_{\mathbb{G}}[b' = b] \leq \Pr_{\mathbb{G}}[r \in L_H^A] \]
Equivalence

\[ \mathcal{E}_{pk}(m) : \]
\[ r \leftarrow \{0, 1\}^\ell; \]
\[ h \leftarrow \{0, 1\}^k; \]
\[ s \leftarrow h \oplus m; \]
\[ c \leftarrow f_{pk}(r) \parallel s; \]
\[ \text{return } c \]

\[ \mathcal{E}_{pk}(m) : \]
\[ r \leftarrow \{0, 1\}^\ell; \]
\[ s \leftarrow \{0, 1\}^k; \]
\[ h \leftarrow s \oplus m; \]
\[ c \leftarrow f_{pk}(r) \parallel s; \]
\[ \text{return } c \]

\[ \equiv \{\top\} \quad G \sim G' \quad \{=_{b,b',r,L_H^A}\} \]

\[ \Pr_G \left[ r \in L_H^A \right] = \Pr_{G'} \left[ r \in L_H^A \right] \]
\[ \Pr_G[ b' = b ] = \Pr_{G'}[ b' = b ] = \frac{1}{2} \]
Equivalence

\[ \mathcal{E}_{pk}(m) : \]
\[ r \leftarrow \{0, 1\}^\ell; \]
\[ h \leftarrow \{0, 1\}^k; \]
\[ s \leftarrow h \oplus m; \]
\[ c \leftarrow f_{pk}(r) \parallel s; \]
\[ \text{return } c \]

\[ \mathcal{E}_{pk}(m) : \]
\[ r \leftarrow \{0, 1\}^\ell; \]
\[ s \leftarrow \{0, 1\}^k; \]
\[ h \leftarrow s \oplus m; \]
\[ c \leftarrow f_{pk}(r) \parallel s; \]
\[ \text{return } c \]

\[ \equiv \{ \top \} \quad \mathbf{G} \sim \mathbf{G}' \quad \{ =_{b,b',r,L_H^A} \} \]

\[ \Pr_{\text{IND-CPA}}[b' = b] - \frac{1}{2} \leq \Pr_{\mathbf{G}'} \left[ r \in L_H^A \right] \]
Reduction

**Game INDCPA**: 

$(sk, pk) \leftarrow \mathcal{K}()$;

$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;

$b \leftarrow \{0, 1\}$;

$c^* \leftarrow \mathcal{E}_{pk}(m_b)$;

$b' \leftarrow \mathcal{A}_2(c^*)$;

return $(b' = b)$

**Encryption** $\mathcal{E}_{pk}(m)$:

$r \leftarrow \{0, 1\}^\ell$;

$s \leftarrow \{0, 1\}^k$;

$c \leftarrow f_{pk}(r) \parallel s$;

return $c$

**Game OW**:

$(sk, pk) \leftarrow \mathcal{K}()$;

$y \leftarrow \{0, 1\}^\ell$;

$y' \leftarrow \mathcal{I}(f_{pk}(y))$;

return $y = y'$

**Adversary** $\mathcal{I}(x)$:

$(m_0, m_1) \leftarrow \mathcal{A}_1(pk)$;

$b \leftarrow \{0, 1\}$;

$s \leftarrow \{0, 1\}^k$;

$c^* \leftarrow x \parallel s$;

$b' \leftarrow \mathcal{A}_2(c^*)$;

$y' \leftarrow \{z \in L_H^A \mid f_{pk}(z) = x\}$;

return $y'$

$\equiv \{\top\} \ G' \sim \text{OW} \left\{ (r \in L_H^A) \langle 1 \rangle \rightarrow (y' = y) \langle 2 \rangle \right\}$

$\Pr_{G'} \left[r \in L_H^A \right] \leq \Pr_{\text{OW}(\mathcal{I})}[y' = y]$
Reduction

**Game INDCPA:**

\[(sk, pk) \leftarrow \mathcal{K}();\]
\[(m_0, m_1) \leftarrow A_1(pk);\]
\[b \leftarrow \{0, 1\};\]
\[c^* \leftarrow \mathcal{E}_{pk}(m_b);\]
\[b' \leftarrow A_2(c^*);\]
return \(b' = b\)

**Encryption \(\mathcal{E}_{pk}(m)\):**

\[r \leftarrow \{0, 1\}^\ell;\]
\[s \leftarrow \{0, 1\}^k;\]
\[c \leftarrow f_{pk}(r) \parallel s;\]
return \(c\)

**Game OW:**

\[(sk, pk) \leftarrow \mathcal{K}();\]
\[y \leftarrow \{0, 1\}^\ell;\]
\[y' \leftarrow \mathcal{I}(f_{pk}(y));\]
return \(y = y'\)

**Adversary \(\mathcal{I}(x)\):**

\[(m_0, m_1) \leftarrow A_1(pk);\]
\[b \leftarrow \{0, 1\};\]
\[s \leftarrow \{0, 1\}^k;\]
\[c^* \leftarrow x \parallel s;\]
\[b' \leftarrow A_2(c^*);\]
\[y' \leftarrow \{z \in L_H^A | f_{pk}(z) = x\};\]
return \(y'\)

\[\models \{\top\} \quad G' \sim \text{OW} \quad \left\{ (r \in L_H^A) \langle 1 \rangle \rightarrow (y' = y) \langle 2 \rangle \right\}\]

\[\Pr_{\text{IND-CPA}(A)}[b' = b] - \frac{1}{2} \leq \Pr_{\text{OW}(\mathcal{I})}[y' = y]\]
Case studies

- Public-key encryption
- Signatures
- Hash designs
- Block ciphers
- Zero-knowledge protocols
- Differential privacy

Based on

- CertiCrypt (2006-2011): Coq based
- EasyCrypt (2009-): SMT based
Limitations

- Abstraction and composition
- Implementation
- Optimization
# Abstraction and composition

## The need

pRHL can capture *instances of* generic proof steps
- Repeated proof effort; requires ability to reason in pRHL
- Does not scale to complex proofs
- Gap between formal proofs and cryptographic practice

## Two mechanisms

- Module system
  - ML-like module system, with negative constraints
  - Allows hybrid arguments
  - Quantification over modules
- Type classes
Modules at work

module type Scheme = {
  fun kg () : skey × pkey
  fun enc(pk:pkey, m:plaintext) : ciphertext
  fun dec(sk:skey, c:ciphertext) : plaintext
}.  

module type ADV = {
  fun choose (pk:pkey) : msg × msg
  fun guess (c:cipher) : bool
}.  

module CPA (S:Scheme, A:ADV) = {
  fun main () : bool = {
    var pk,sk,m_0,m_1,b,b',challenge;
    (pk,sk) = S.kg();
    (m_0,m_1) = A.choose(pk);
    b $\leftarrow \{0,1\};$
    challenge = S.enc(pk, b?m_1:m_0);
    b' = A.guess(challenge);
    return b' = b;
  }
}.  

Reduction argument:

\forall (A <: Adv), \exists (I <: Inverter), Pr[CPA(A) : res] - 1/2 \leq Pr[OW(I) : res].

Existential quantification over modules would require support for complexity claims
Implementation

Painful gap between provable security and real world

- Proofs reason about algorithmic descriptions
- Standards constrain implementations
- Attackers target executable code

Real-world crypto is breakable; is in fact being broken; is one of many ongoing disaster areas in security. Bernstein, 2013

Reasoning about implementations

- C-mode
- Use CompCert as a backend
- Account for side-channel countermeasures
OAEP: real-world security

- RSA implementations are vulnerable to timing attacks
- PKCS1.5 vulnerable to padding oracle attacks
- RSA is a permutation only on strict subset of $[0..2^k]$: careless conversion leads to timing attacks
Implementation of OAEP

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

Decryption \( D_{\text{PKCS-C}}(sk)(res, c) : \)
\[
\text{if } (c \in \text{MsgSpace}(sk)) \text{ then}
\]
\[
\{ (b0, s, t) \leftarrow f^{-1}_{sk}(c);
\]
\[
h \leftarrow MGF(s, hL); i \leftarrow 0;
\]
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 \((i < dbLen)\)
\[
\{ p[i] \leftarrow s[i] \oplus g[i]; i \leftarrow i + 1; \}
\]
\[
l \leftarrow \text{payload\_length}(p);
\]
if \((b0 = 0^8 \land [p]^{hLen} = 0..01 \land
\]
\[
[p]^{hLen} = LHash)\)
then
\[
\{rc \leftarrow \text{Success};
\]
\[
\text{memcpy}(res, 0, p, dbLen - l, l); \}
\]
else \[
\{rc \leftarrow \text{DecryptionError}; \}
\]
else \[
\{rc \leftarrow \text{CiphertextTooLong}; \}
\]
return \(rc\);
Provable security of C and executable code

C-mode for EasyCrypt

- Prove PKCS using C-mode for EasyCrypt: arrays are base-offset representation and match subset of C arrays (no aliasing or overlap possible, pointer arithmetic only within an array)
- Carry proof from 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

- Account for some side-channels

Certified compilers

Reduction proof rests on semantic preservation. Use CompCert
CompCert (Leroy, 2006)

- Optimizing C compiler implemented in Coq
- Formal proof of semantic preservation
Application to OAEP

### Baseline security
- idealized operations moved to the environment
  - random sampling of bitstrings
  - hash function (random oracle)
- “trusted-lib” mechanism for arithmetic libraries

### PC security
- annotation mechanism is used to track control flow
- ASM analyser checks that compiler does not add branches
- “trusted-lib” must also verify leakage assumptions
Implementations and performance

- Carefully craft C code to avoid hidden branching (operations on bitstrings or booleans)
- Compile with CompCert
- Check no new branch was introduced
- Links generated code with LIP (secure in PC model)
Cache-based side-channel attacks

A recipe for security disaster
- Array accesses with high indices
- Adversary gains knowledge through cache
- Lead to devastating cache-based attacks: AES, BlowFish, DES, RC4, Snow...
- Alternative: constant-time crypto

Protection mechanism
- alias and information flow analysis (CompCert assembly)
- tag arrays accessed with secret indices as “stealth”
- typable programs w/o “stealth” address are constant-time
- typable programs execute securely on stealth memory (provided stealth addresses are mapped correctly)
<table>
<thead>
<tr>
<th>Example</th>
<th>LOC</th>
<th># Stealth Addresses</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>744</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>DES</td>
<td>836</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Blowfish</td>
<td>279</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>RC4</td>
<td>164</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>Snow</td>
<td>757</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Salsa20</td>
<td>1077</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TEA</td>
<td>70</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SHA256</td>
<td>419</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Automated synthesis of 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)
An algebraic view of padding-based schemes

Encryption algorithms are modelled as algebraic expressions

\[ E ::= m \quad \text{input message} \]
\[ 0 \quad \text{zero bitstring} \]
\[ \mathcal{R} \quad \text{uniform random bitstring} \]
\[ E \oplus E \quad \text{xor} \]
\[ E \parallel E \quad \text{concatenation} \]
\[ [E]_s \quad \text{projection} \]
\[ H(E) \quad \text{hash} \]
\[ f(E) \quad \text{trapdoor permutation} \]

Decryption algorithms are modelled in a mild extension
Semantics

Left-to-right evaluation with sharing, yields a pWHILE procedure

Example

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

interpreted as:

$$r \leftarrow \{0, 1\}^k;$$
$$g \leftarrow G(r);$$
$$s \leftarrow g \oplus (m \| 0);$$
$$h \leftarrow H(s);$$
$$\text{return } f_{pk}(s \| (h \oplus r))$$
Approach
Attack finding

- Based on standard symbolic tools
  - deducibility and static equivalence
- For efficiency, first apply simple filters, eg
  - is decryption possible without a key? \( m \parallel f(r) \)
  - is encryption randomized? \( f(m) \)
  - is randomness extractable without a key? \( r \parallel f(m \oplus r) \)
**Chosen-plaintext security: principles**

<table>
<thead>
<tr>
<th>Failure event</th>
<th>Replace $H(e)$ by fresh $r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimistic sampling</td>
<td>Replace $e \oplus r$, where $r$ is fresh, by $r$</td>
</tr>
<tr>
<td>Probability</td>
<td>Compute probability of $b = b'$ or $e \in L$</td>
</tr>
<tr>
<td>Reduction</td>
<td>Find inverter and apply one-wayness</td>
</tr>
</tbody>
</table>
Chosen-plaintext security: formalism

Judgment

\[ c \vdash p \varphi \]

- Reasons about probability of events
- Concrete probability can be computed
  - on the fly
  - a posteriori (judgments use only \( p = 0, \frac{1}{2} \))
- Side-conditions are discharged by symbolic methods
  - what is the entropy of \( e \)?
  - can I compute \( e' \) from \( e \)?
Chosen-plaintext security: proof rules

\[ m \not\in c^* \]
\[ \frac{c^* : \frac{1}{2} \text{Guess}}{} \text{[Indep]} \]

\[ e \vdash \vec{r} \quad \vec{r} \cap R(c^*) = \emptyset \]
\[ \frac{c^* : 0 \; e \in L_H^A}{e \vdash A \vec{r} \left\| \vec{r}_0 \right\| m \vdash A c^* \quad \vec{r} \cap \vec{r}_0 = \emptyset} \text{[OW]} \]
Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Proof</th>
<th>Attack</th>
<th>Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPA</td>
<td>116433</td>
<td>901929</td>
<td>12627</td>
</tr>
<tr>
<td>CCA</td>
<td>28001</td>
<td>182730</td>
<td>66332</td>
</tr>
</tbody>
</table>

Remarks:
- 16185 out of 66332 CCA unknown non-redundant
- CPA unknown seem secure
- Proving completeness seems very hard
- Found (by inspection) one new and interesting scheme
  \[ f(r \parallel m \oplus G(r)) \]
- Missing methods for mining the database
Some ongoing projects

- Foundations and automation
- Case studies: AKE, MPC, faults, verifiable computation
- Synthesis of dlog and pairing-based encryption
- Verified batch verifiers
- Automated analysis in (multi-linear) generic groups
Conclusion

- Solid foundation for cryptographic proofs
- Formal verification of emblematic case studies
- Narrowing the gap between proofs and code
- Automated analysis and synthesis of crypto schemes

http://www.easycrypt.info
EasyCrypt toolchain

- ZooCrypt
- AutoBatch
- ...
- CertiCrypt
- CompCert
- StealthCert

User → EasyCrypt → Why3