ANR ProSe

Security protocols: formal model, computational model, and implementations

Bruno Blanchet
bruno.blanchet@inria.fr

INRIA Paris-Rocquencourt

November 2012
ANR ProSe: participants

- **INRIA Paris-Rocquencourt (EPI Prosecco and EPI Cascade)**
  - Permanent: Bruno Blanchet, David Pointcheval, Graham Steel (Jan. 2012–)
  - Post-docs: Gergei Bana (Feb. 2012–), Ben Smyth (July 2012–)
  - PhD students: David Cadé, Miriam Paiola

- **Laboratoire Spécification et Vérification (LSV, ENS Cachan)**
  - Post-docs: Rohit Chadha (–June 2012), Joe-Kai Tsay (–Aug. 2011)
  - PhD students: Vincent Cheval (–Jan. 2013), Guillaume Scerri (Sept. 2011–)

- **INRIA Nancy Grand-Est (EPI Cassis)**
  - Permanent: Steve Kremer (Sept. 2011–), Mathieu Turuani
  - Intern: Apoorvaa Deshpande (June–Nov. 2012)

- **VERIMAG**
  - Permanent: Yassine Lakhnech, Pierre Corbineau, Cristian Ene, Pascal Lafourcade
  - PhD students: Jannik Dreier, Mathilde Duclos, Marion Daubignard (–Jan. 2012)
Security protocols use cryptography to secure communications over insecure networks.

- Securing communications becomes more and more important:
  - Mobile sensor networks
  - Mobile terminals (smartphones, . . .)
  - Cards (electronic passports, cash cards, . . .)

- Securing communications is difficult:
  - Design of security protocols error-prone
  - Errors not detected by testing

⇒ need for formal methods.
Security protocol verification

Three levels:

- **The symbolic level**: a formal, abstract model.
  - The cryptographic primitives are ideal blackboxes.
  - Messages are terms on these primitives.
  - The adversary uses only those primitives.
  - Proofs can be done automatically.

- **The computational level**: a realistic model.
  - Messages are bitstrings.
  - The cryptographic primitives are functions on bitstrings.
  - The adversary is a polynomial-time Turing machine.
  - Proofs done manually by cryptographers.

- **The implementation level**: the program running the protocol.

Our goal: provide proofs/find attacks at all three levels.
Existing approaches to automating symbolic proofs

- Many tools exist for verifying protocols in the symbolic model (AVISPA, FDR, Scyther, ...)
- One tool we build upon is ProVerif:
  - Fully automatic symbolic protocol verifier.
  - Works for unbounded number of sessions and message space.
  - Handles a wide range of cryptographic primitives, defined by rewrite rules or equations.
  - Handles various security properties: secrecy, authentication, a limited class of equivalences.
  - Limitations: does not always terminate and may answer “I don’t know”.
Existing approaches to automating computational proofs

- **Computational soundness:**
  Security in the symbolic model $\Rightarrow$ computational model modulo additional assumptions.

- **Direct computational proofs:**
  Automate proofs by sequences of games
  - CryptoVerif
    - tool that generates proofs by sequences of games
    - has automatic and manual modes
  - CertiCrypt/EasyCrypt
    - Framework for machine-checked cryptographic proofs in Coq
    - Good for proving primitives: can prove complex mathematical theorems, but with a lot of human effort
    - Improved by EasyCrypt: generates CertiCrypt proofs from proof sketches (sequence of games and hints)
Task 2: Simple, automated proofs with strong, computational guarantees

1. Extensions of **symbolic methods**:
   - Group protocols
   - Observational equivalence
   - Extending attacker capabilities: length of messages, ...

2. Extensions of **soundness proofs**:
   - Dishonest keys
   - Exclusive or
   - Other primitives: blind signatures, re-encryption
   - Weakening security assumptions

3. Extensions of **direct computational proofs**:
   - Diffie-Hellman key agreement
   - Exclusive or

4. Case study

5. Comparison of the approaches
Task 3: Proved implementations of protocols

1. Generation of implementations from CryptoVerif specifications
   - Implementation of the compiler CryptoVerif to OCaml
   - Proof of the compiler
   - Extensions of the input language
   - Case study

2. Hoare style proofs for cryptographic primitives
   - Developing a Hoare style verification method
   - Case study
Modularity is key to scale up:

- Computational soundness proofs generally hold for a fixed set of primitives, a fixed security property, a fixed protocol model.
- They become more and more complex when more primitives are added.

⇒ need for modular computational soundness proofs.
Task 5: Logic for computational indistinguishability

Logic to reason about indistinguishability in the frame of exact security:

- Define a model of cryptographic systems
- Develop a formal system for proving protocols in this model
- Develop a proof search algorithm
- Formalize and implement the logic in a theorem prover (Coq)
Task 2, highlight: Towards unconditional soundness: computationally complete symbolic attacker

LSV, INRIA Paris-Rocquencourt, LORIA

Usually:

- in *symbolic models*, we specify what the *attacker can do*, e.g. apply encryption, decryption, signatures, . . .
- in *computational models*, we specify what the *attacker cannot do*, e.g. cannot distinguish two ciphertexts, cannot forge signatures, . . .

⇒ difficult to get computational soundness.

**Main idea:** design a new *symbolic model*, in which we specify what the attacker cannot do.
New symbolic model

Model of terms, in first order logic.
Predicates:

- $t_1 = t_2$ is equality.
- $t_1, \ldots, t_n \vdash t$ means that $t$ can be computed from $t_1, \ldots, t_n$.

The logic has two semantics:

- **Symbolic model:** the interpretation of these predicates not fixed; they will be defined by axioms, specifying things that the attacker cannot do.

- **Computational model:** the interpretation of these predicates is fixed. Intuitively, they are true when they hold up to negligible probability in the computational model.

(The real semantics is more complicated.)
More predicates

The following predicates are defined in both models:

- $\hat{\phi}, t_1, \ldots, t_n \vdash t$: $t$ can be computed from $t_1, \ldots, t_n$ and the messages sent so far (defined from $t_1, \ldots, t_n \vdash t$).
- randgen($s$): $s$ is chosen randomly.
- $t \sqsubseteq \hat{\phi}$: $t$ is a subterm of the messages sent so far.
Examples of axioms

- Function of derivable terms:

\[
\hat{\phi} \vdash t_1 \land \ldots \land \hat{\phi} \vdash t_n \Rightarrow \hat{\phi} \vdash f(t_1, \ldots, t_n)
\]

- The attacker cannot guess a fresh, randomly generated key \( k \) (except with negligible probability):

\[
\text{randgen}(k) \land k \not\sqsubseteq \hat{\phi} \Rightarrow \neg(\hat{\phi} \vdash k)
\]

- If the encryption scheme \( \{\cdot\}_e^R \) is IND-CCA2, then

\[
\forall t, \forall K, \forall R, \left( \text{randgen}(K) \land \text{randgen}(R) \land R \not\sqsubseteq \hat{\phi} \land \hat{\phi}, \{t\}_e^R \vdash t \Rightarrow dK \sqsubseteq \hat{\phi} \lor \hat{\phi} \vdash t \right)
\]
The axioms must be proved computationally sound. (They hold up to negligible probability in the computational model.)

Theorem (Computational soundness [Bana, Comon-Lundh, POST’12])

For a bounded number of sessions, if there is a computational attack, then there is also a symbolic attack.
Further results in this line of research

- Application to a proof of the Needham-Schroeder protocol in the computational model \[\text{[Bana, Adao, Sakurada, FSTTCS'12]}\]
- A decision procedure for the new symbolic model, for a bounded number of sessions \[\text{[Comon-Lundh, Cortier, Scerri, FCC’12]}\]
- Computational completeness result for this logic (completeness of the axiomatisation of $\vdash$)
- Extension to key usability
Task 2: group protocols

For a class of protocols with lists, we have shown that

ProVerif proves secrecy for lists of length 1 ⇒ secrecy holds for lists of any length

[Paiola, Blanchet, POST’12]

Main constraint: the protocol needs to manipulate all elements of lists uniformly.

Application: secrecy of the session key in the Asokan-Ginzboorg group protocol.

We are working on an extension of the resolution algorithm of ProVerif to support protocols with lists in which all elements of the list are not manipulated uniformly.
Task 2: observational equivalence LSV, INRIA Paris-Rocquencourt

- **Decision procedure** for trace equivalence, extending previous results to protocols with
  - conditional branches,
  - non-determinism coming from private channels

  [Cheval, Comon-Lundh, Delaune, CCS’11]

- **Modularity:** We have given sufficient conditions so that
  privacy properties of components ⇒ privacy properties of the composed protocol

  (Privacy properties are expressed as trace equivalences.)

  [Arapinis, Cheval, Delaune, CSF’12]

- **Extension of ProVerif:**
  - ProVerif can prove equivalences between processes that have the same structure, but differ by the terms (messages) they contain.
  - We have extended terms in order to express “if then else” inside terms.
  - This extends the class of equivalences that can be proved.

  [Cheval, Blanchet, submitted]
Most computational soundness results do not allow dishonest keys: all keys must be certified by a trusted authority.

This is not realistic for symmetric encryption.

We lifted this assumption by generalizing the symbolic model.

[Comon-Lundh, Cortier, Scerri, POST’12]
We have used the extension of CryptoVerif to support the Computational Diffie-Hellman assumption (CDH) in two case studies:
- the One-Encryption Key Exchange protocol (OEKE), a password-based protocol
  [Blanchet, CSF’12]
- the Secure SHell protocol (SSH)
  [Cadé, Blanchet, ARES’12]

We have also extended CryptoVerif to support the Decisional Diffie-Hellman assumption (DDH).
Goal: obtain implementations of protocols proved secure in the computational model.

Idea:
- prove the protocol specification secure in the computational model using CryptoVerif,
- compile the specification into an implementation, using a specific compiler.
From CryptoVerif specifications to implementations

CryptoVerif specification → Our Compiler → Protocol Code → OCaml Compiler → Implementation → Proof in the computational model

Caption: Tool Input Result
We have implemented the compiler, and used it to generate an implementation of the Secure SHell (SSH) transport layer protocol. [Cadé, Blanchet, ARES’12]

We have proved that this compiler preserves security:
the specification satisfies a security property ⇒ the generated implementation satisfies the same security property

[Cadé, Blanchet, submitted]

Compiler available as part of CryptoVerif, at http://cryptooverif.inria.fr.
Task 3: Hoare style proofs for cryptographic primitives

Automatically proving security of cryptographic primitives

1. Modeling security properties
2. Defining a language
3. Building an Hoare Logic
4. Proving the security
Task 3: Hoare style proofs for cryptographic primitives

Automatically proving security of cryptographic primitives

1. Modeling security properties
2. Defining a language
3. Building an Hoare Logic
4. Proving the security

FOR

- extension to exact security for
  Generic Asymmetric Encryption Schemes
  and Generic Symmetric Encryption Modes
- Generic Message Authentication Codes

[Gagné, Lafourcade, Lakhnech, submitted]
Task 4: Compose and decompose protocols and proofs: get modular proofs of security

- We have defined **cryptographic games** that can serve in defining cryptographic primitives, and that imply computational soundness results.
  - These games are independent of the underlying execution model.
  - They should allow us to obtain combination results more easily.
  ⇒ more **modular proofs**.

- We are currently proving that these games imply the **trace mapping** property in a very general execution model.

- The next steps will be to show that
  - to show that these definitions are realizable under standard assumptions on cryptographic primitives,
  - and to prove combination results.
Task 5, highlight: Logic for computational indistinguishability

- Most security criteria and proofs in the computational model rely on the concept of indistinguishability.
- ⇒ CIL, Computational Indistinguishability Logic.
  - reason on indistinguishability
  - directly in the computational model
  - additional assumptions can be plugged in (e.g. ROM)
CIL: the model (informally)

Scenario of the interaction:

**Adversary** $A$ | **Oracle System** $\mathcal{O}$
---|---
initialize |
values |
query |
answer |
\[ \ldots \] | query |
answer |
sol to challenge |
done |

Oracle system $\mathcal{O} = \text{set of oracles.}$
CIL: judgments in the logic

Logic

1. **judgments**: statements on which we reason, types of conclusions
2. **19 rules**: sound reasoning steps

1. **Is-bounded-by**, denoted $\mathcal{O} : \epsilon \ E$
   
   For every adversary $\mathcal{A}$, the probability that $\mathcal{A} \mid \mathcal{O}$ satisfies $E$ is at most $\epsilon(\mathcal{A})$
   
   $$\Pr[\mathcal{A} \mid \mathcal{O} : E] \leq \epsilon(\mathcal{A})$$

2. **Is-indistinguishable-from**, denoted $\mathcal{O} \sim_{\epsilon} \mathcal{O}'$
   
   For every adversary $\mathcal{A}$, the probability that $\mathcal{A}$ distinguishes $\mathcal{O}$ from $\mathcal{O}'$ is at most $\epsilon(\mathcal{A})$. 
CIL: basic examples of rules

1. $\sim$ is an equivalence relation:

\[
\begin{align*}
\emptyset & \sim_0 \emptyset \\
\emptyset & \sim_\varepsilon \emptyset \\
\emptyset & \sim_\varepsilon \emptyset \\
\emptyset & \sim_{\varepsilon + \varepsilon'} \emptyset''
\end{align*}
\]

2. Union rule:

\[
\begin{align*}
\emptyset : \varepsilon_i E_i \ (i \in I) & \Rightarrow \bigvee_{i \in I} E_i \\
\emptyset : \sum_{i \in I} \varepsilon_i E & \Rightarrow UR
\end{align*}
\]
CIL: examples of rules using bisimulation up to

\[\begin{align*}
\emptyset : \epsilon F \neg \varphi & \quad \emptyset \equiv_{R, \varphi} \emptyset' \\
\emptyset \sim_{\epsilon} \emptyset' & \quad \text{l-Bis}
\end{align*}\]

Intuition: if \(\varphi\) holds, \(\emptyset\) and \(\emptyset'\) behave the same \((\emptyset \equiv_{R, \varphi} \emptyset')\), so that they are distinguishable only if \(\neg \varphi\) ever happens \((F \neg \varphi)\).

\[\begin{align*}
\emptyset : \epsilon F \neg \varphi & \quad \emptyset \equiv_{R, \varphi} \emptyset' \\
\emptyset' : \epsilon F \neg \varphi & \quad \text{B-Bis U}
\end{align*}\]

Intuition: if \(\varphi\) holds, \(\emptyset\) and \(\emptyset'\) behave the same, so that the probability of going in a state where \(\varphi\) becomes false is equal in both systems.
CIL: results

- Formalization and **soundness proof in Coq**
  [Corbineau, Duclos, Lakhnech, CPP’11]
- Proof of a key-exchange protocol resistant to some side-channel attacks
  [Barthe, Duclos, Lakhnech, FPS’11]
- Modularity result to verify iterative **hash function constructions**. Applied to show indifferentiability of SHA-3 candidates. Found an error in the indifferentiability proof of one candidate.
  [Daubignard, Fouque, Lakhnech, CSF’12]
- In progress:
  - Integration of CIL in EasyCrypt
  - Automation of proofs in EasyCrypt
The work basically follows the expected schedule, with minor changes.

**Initial schedule:**

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
<th>Task 4</th>
<th>Task 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2.2. Computational soundness</strong></td>
<td><strong>2.3. Extensions of CryptoVerif</strong></td>
<td><strong>3.1.2. Extensions of the specification language</strong></td>
<td><strong>4.2. Relevance of the games w.r.t. assumptions</strong></td>
<td><strong>5.2. Indistinguishability logic</strong></td>
</tr>
<tr>
<td><strong>2.4 Case study: FOO</strong></td>
<td></td>
<td><strong>3.1.3. Proof of implementations of Kerberos, TLS, FOO</strong></td>
<td><strong>4.3. Relevance of the games w.r.t. soundness proofs</strong></td>
<td><strong>5.3. Heuristics for proof search</strong></td>
</tr>
<tr>
<td><strong>2.5 Comparison of the approaches</strong></td>
<td></td>
<td></td>
<td><strong>4.4. Modularity results</strong></td>
<td><strong>5.4. Implementation on top of a theorem prover</strong></td>
</tr>
</tbody>
</table>
The work basically follows the expected schedule, with minor changes.

Updated:

Task 1
- 1. Project coordination
- 2.1. Symbolic model
- 2.2. Computational soundness
- 2.3. Extensions of CryptoVerif

Task 2
- 2.4 Case study: FOO
- 2.5 Comparison of the approaches

Task 3
- 3.1. Generation of implementations from CryptoVerif specifications
  - 3.1.1. Foundations
  - 3.1.2. Extensions of the specification language
  - 3.1.3. Case studies: SSH, ...

Task 4
- 3.2. Hoare style proofs for cryptographic primitives
  - 3.2.1. Hoare style verification method
  - 3.2.2. Case study on primitives

Task 5
- 4.1. Deducibility/indistinguishability games
- 4.2. Relevance of the games w.r.t. soundness proofs
- 4.3. Relevance of the games w.r.t. assumptions
- 4.4. Modularity results
- 4.5 A computational proof of FOO

Task 6
- 5.1. General model for cryptographic systems
- 5.2. Indistinguishability logic
- 5.3. Heuristics for proof search
- 5.4. Implementation on top of a theorem prover

Task 7
- 6. Comparison of approaches
Deliverables (1)

Task 1
- T0+6: Progress report (05/09/2011)
- T0+18: Progress report (15/07/2012)
- T0+30: Progress report
- T0+48: Final report

Task 2
- T0+12: Synthesis on protocol verification (27/01/2012)
- T0+12: Version of CryptoVerif that handles DDH (27/01/2012)
- T0+24: Prototype implementation for task 2.1.2
- T0+36: Version of CryptoVerif that handles XOR
- T0+48: Results of our tools on benchmarks

Task 3
- T0+12: Compiler from specifications to implementations (27/01/2012)
- T0+24: Paper on the generation of proved implementations
- T0+24: Paper on the verification of cryptographic primitives
- T0+48: Enhanced compiler from specifications to implementations
- T0+48: Results of our tools on benchmarks
Deliverables (2)

- Task 4
  - T0+12: Deducibility/indistinguishability games (27/01/2012)
  - T0+24: Relevance of the games
  - T0+36: Composability results
  - T0+48: Application to a computational proof of FOO

- Task 5
  - T0+24: First prototype implementation
  - T0+36: Formalized proof of IND-CCA security of OAEP
  - T0+48: Enhanced prototype implementation
Collaboration

- Collaboration in particular through joint students:
  - Vincent Cheval (LSV, INRIA Paris-Rocquencourt)
  - Guillaume Scerri (LSV, LORIA)
  - Apoorvaa Deshpande (LORIA, LSV)

- Several joint projects:
  - Computationally-complete symbolic attacker (LSV, INRIA Paris-Rocquencourt, LORIA)
  - Proof of observational equivalences (LSV, INRIA Paris-Rocquencourt)
  - Modular computational soundness proofs (LORIA, LSV)
  - Verification of implementations: Tasks 3.1 (INRIA Paris-Rocquencourt) and 3.2 (VERIMAG) are complementary
Project communication

- Meetings:
  - Kick-off meeting: Monday, February 28, 2011, ENS Ulm
  - Workshop FCC: June 30, 2011, IHP
  - Joint ProSe/SCALP meeting, January 12-13, 2012, VERIMAG
    (Marion Daubignard’s PhD defense and Workshop on Computer-Aided Security)
  - Meeting: Thursday, October 25, 2012, INRIA Place d’Italie
  - FCC’11: 3 PC members and 2 talks from ProSe
  - FCC’12: 3 PC members and 4 talks from ProSe.
- 4 ProSe members in the informal FCC steering committee.
Publications of the ProSe project:

- 1 book chapter
- 9 international conferences with programme committee and proceedings (CCS, CSF, FSTTCS, POST, ...)
- 1 workshop with programme committee and post-proceedings

Software

- **CryptoVerif**, [http://cryptoverif.inria.fr](http://cryptoverif.inria.fr)
  - Compiler from specifications to implementations
  - Extensions, in particular to Diffie-Hellman key agreements
- **ProVerif**, [http://proverif.inria.fr](http://proverif.inria.fr)
  - Extensions to prove more observational equivalences.
- **AKiSs (verifier for trace equivalence)**, [https://github.com/ciobaca/akiss](https://github.com/ciobaca/akiss)
- **Hoare logic verifier for MACs**, [http://www-verimag.imag.fr/~gagne/macChecker.html](http://www-verimag.imag.fr/~gagne/macChecker.html)
Conclusion

- Progress in **all directions**:
  - Extensions of existing methods for *symbolic* and *computational* proofs.
  - New methods for *computational* proofs:
    - Bana–Comon-Lundh logic for computational soundness
    - Modular computational soundness proofs
    - Computational indistinguishability logic
  - New methods for obtaining **proved implementations**:
    - Generation of implementations from CryptoVerif specifications
    - Hoare-style proofs for cryptographic primitives

- Both **theoretical** and **practical** impact:
  - Publications
  - Freely available software
The planned objectives should be reached by the end of the project.

- This project will provide considerable progress in the field of protocol verification.

In the longer term, make our work and tools well-known and used by others.

- Contacts at DGA, MSR, …

Some aspects will still need investigation after the end of the project:

- More work on the proof of implementations
- Side-channels attacks