FormaCrypt: Formal Computational Cryptography

Bruno Blanchet[1], David Monniaux[1], David Pointcheval[1]
Jean Goubault-Larrecq[2], Mathieu Baudet[2], Steve Kremer[2]
Véronique Cortier[3], Mathieu Turuani[3], Bogdan Warinshi[3]
Martín Abadi[4]

[1]LIENS [2]LSV [3]LORIA [4]UCSC & Microsoft Research

November 2006

ARA SSIA FormaCrypt: participants

- Laboratoire d'Informatique de l'Ecole Normale Supérieure (LIENS)
  - Bruno Blanchet
  - David Monniaux
  - David Pointcheval
- Laboratoire Spécification et Vérification (LSV), ENS Cachan
  - Jean Goubault-Larrecq
  - Mathieu Baudet (leaving to DCSSI)
  - Steve Kremer
  - Laurent Mazaré (starting October 1st, 2006)
- Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA)
  - Véronique Cortier
  - Mathieu Turuani
  - Bogdan Warinschi
- Scientific advisor: Martín Abadi

Proofs of cryptographic protocols

There are two main frameworks for analyzing security protocols:

- The Dolev-Yao model: a formal, abstract model.

  The cryptographic primitives are ideal blackboxes.
  The adversary uses only those primitives.

  Proofs can be done automatically.

- The computational model: a realistic model.

  The cryptographic primitives are functions on bit-strings.
  The adversary is a polynomial-time Turing machine.

  Proofs are done manually.

Our goal: bridge the gap between these two frameworks.

Three approaches

We have considered three approaches:

- Direct approach: build an automatic computationally sound prover.
- Intermediate approach: design a computationally sound logic, for reasoning symbolically on protocols.
- Modular approach: obtain computational soundness results, that is, show that security in the formal model implies security in the computational model.

We will obviously compare these approaches on examples ranging from protocols of the literature to more complex, realistic protocols.

An automatic computationally sound prover

We have implemented an automatic prover sound in the computational model:

- proves secrecy properties and that events can be executed only with negligible probability.
- handles various cryptographic primitives: MACs (message authentication codes), stream and block ciphers, public-key encryption, signatures, hash functions, ...
- works for a parametric number of sessions with an active adversary.
- gives a bound on the probability of an attack (exact security).

Produced proofs

As in Shoup's or Bellare and Rogaway's method, the proof is a sequence of games:

- In the first game, the adversary plays against the real protocol.
- One goes from one game to the next by syntactic transformations or by applying security assumptions on cryptographic primitives.

  The difference of probability between consecutive games is bounded.
- The last game is "ideal": the security property can be read directly on it.

  (The advantage of the adversary is 0 for this game.)

Games are formalized in a process calculus.

Input and output of the prover

Our prover is given as input

- the security assumptions on the cryptographic primitives;
- the initial game, given in a syntax close to the standard notations in cryptography;
- the properties to prove.

The prover outputs

- the (negligible) probability that each desired property is wrong;
- the sequence of games that leads to the proof;
- a succinct explanation of the transformations performed between games.

The user is allowed (but does not have) to interact with the prover to make it follow a specific sequence of games.

Experimental results

The prover is available at:
http://www.di.ens.fr/~blanchet/cryptoc/

We have tested it successfully on many examples:

- protocols of the literature: incorrect and corrected versions of Otway-Rees, Yahalom, Needham-Schroeder shared-key and public-key, and Denning-Sacco public key;
- the Full Domain Hash signature scheme;
- encryption schemes of [Bellare and Rogaway, CCS'93].

We plan to test it on more examples in the future.

Planed extensions

1. Prove correspondence properties of the form: if some event has been executed, then other events have been executed before (except in cases of negligible probability).
   These properties are useful for proving authentication.
2. Handle other primitives, such as Diffie-Hellman key agreements.
3. Improve the proof strategy, for more automation.

A computationally sound logic

We have studied the following approach for proving protocols:

1. start from an existing protocol logic, designed in the formal model (here, the Protocol Composition Logic),
2. and adapt it to the computational model.

Advantage of this approach:
proofs that use the logic in the formal model can be adapted to the computational model with only minor corrections.

The Protocol Composition Logic (PCL)

The Protocol Composition Logic allows symbolic reasoning on protocols:

- The protocol is specified in a simple "protocol programming language".
- The logic consists of both
  - logical formulas (including predicates that specify knowledge and actions of participants)
  - and modal formulas, similarly to the Floyd-Hoare logic (if a formula is true at some point and certain actions are executed, then another formula holds afterwards).
- The logic allows compositional reasoning.

Adapting PCL to the computational model

We have adapted the Protocol Composition Logic to the computational model:

- We have given a new probabilistic, polynomial-time semantics to the logic: a formula is true if it holds with asymptotically overwhelming probability.
- We have given a meaning to logical connectives in this semantics.
- We had to extend the logic with new predicates that make sense in the computational model but not in the formal one, for example to express indistinguishability.

A soundness proof has been done for a subset of PCL with positive indistinguishability tests.

Proving more complex properties

We have extended this logic to more complex properties, in particular secrecy of keys.

This result allows the following compositional reasoning:

- we first prove the security of keys established using a key exchange protocol;
- then, we infer the security of a secure channel application that uses these keys.

Planed extensions

1. Soundness for any proof in PCL extended with computational tests.
2. Make the semantics more direct and natural.

Formal model: several abstractions

Messages are modeled by terms

- $\{m\}_k$: message m encrypted by k
- $\langle m_1, m_2 \rangle$: pair of m_1 and m_2
- ...

$\longrightarrow$ no collisions:

$$\forall m, m', k, k' \quad \{m\}_k \neq \{m'\}_{k'}, \{\{m\}_k\}_k \neq m, \langle m, m' \rangle \neq \{m\}_k, \ldots$$

Perfect encryption assumption:

Nothing can be learned from $\{m\}_k$ except if k is known.

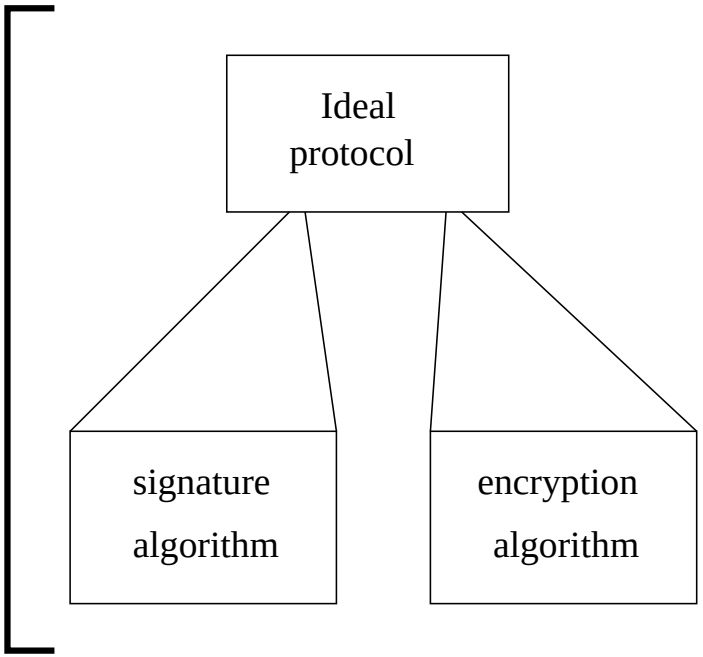
Much easier to analyze automatically

- numerous decidability results
- numerous automatic tools

Goal: soundness of the formal model

Composition of two approaches

Implemented protocol



Formal approach: verification of idealized protocols

Cryptographers: verification of the cryptographic primitives

Combination result in presence of hash functions

Example

$A \to B : h(s)$

s is symbolically secret but not indistinguishable to an attacker: $h(n_b), n_0, n_1 \to b$

Results:

1. Design of a new formal secrecy property
2. Proof of its soundness and its faithfulness w.r.t. indistinguishability in our new setting:
   - pairing
   - asymmetric encryption
   - hashes (random oracle model)
3. NP-completeness of the secrecy property

Extension to symmetric encryption

Cycles of the form $\{k_1\}_{k_2}$, $\{k_2\}_{k_3}$, $\{k_3\}_{k_1}$ must be forbidden.

As a preliminary result, we obtain that:

> *Deciding if an intruder can create key cycles on keys of honest agents is NP-complete for a bounded number of sessions.*

Guessing attacks

Guessing attacks: The adversary can guess low entropy values such as passwords and verify them off-line.

Results:

1. Design of a new security property for the interaction between normal and password-based encryption
2. Proof of soundness of static-equivalence (passive case) w.r.t. indistinguishability for:
   - asymmetric encryption
   - symmetric encryption
   - password-based encryption

Planed extensions

1. Study branching properties, such as fairness.
2. Prove the secrecy of keys (not only of nonces).
3. Extend the results to protocols that use primitives with more complex equational theories
   (Diffie-Hellman, XOR, CBC encryption).

Conclusion

- We have investigated three different approaches for bridging the gap between the computational and the formal models of cryptography.
- Up to now, the project has produced 8 papers and 2 tools that provide proofs of protocols in the computational model.
- These three approaches will be extended and compared in the next years.

More details, publications, and software available at:
http://www.di.ens.fr/~blanchet/formacrypt/