CryptoVerif: state of the art, perspectives, and relations to other tools

Bruno Blanchet

INRIA Paris
Bruno.Blanchet@inria.fr

April 2017

Outline

1. CryptoVerif
2. Future directions
3. Relations to other tools:
   - ProVerif
   - EasyCrypt
   - Scary

CryptoVerif, http://cryptoverif.inria.fr/

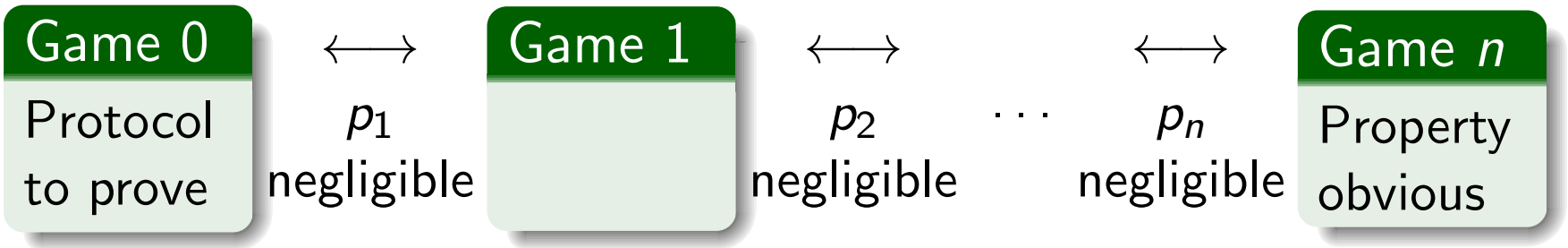
CryptoVerif is a semi-automatic prover that:

- works in the computational model.
- generates proofs by sequences of games.
- proves secrecy and correspondence properties.
- provides a generic method for specifying properties of cryptographic primitives which handles MACs (message authentication codes), symmetric encryption, public-key encryption, signatures, hash functions, Diffie-Hellman key agreements, ...
- works for N sessions (polynomial in the security parameter), with an active adversary.
- gives a bound on the probability of an attack (exact security).

Proofs by sequences of games

CryptoVerif produces proofs by sequences of games, like those of cryptographers [Shoup, Bellare&Rogaway]:

- The first game is the real protocol.
- One goes from one game to the next by syntactic transformations or by applying the definition of security of a cryptographic primitive. The difference of probability between consecutive games is negligible.
- The last game is "ideal": the security property is obvious from the form of the game.
  (The advantage of the adversary is 0 for this game.)



Input and output of the tool

1. Prepare the input file containing
   - the specification of the protocol to study (initial game),
   - the security assumptions on the cryptographic primitives,
   - the security properties to prove.
2. Run CryptoVerif
   - Automatic proof strategy or manual guidance.
3. CryptoVerif outputs
   - the sequence of games that leads to the proof,
   - a succinct explanation of the transformations performed between games,
   - an upper bound of the probability of success of an attack.

Main case studies

- Kerberos V, with and without PKINIT (with Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay).
- OEKE (variant of Encrypted Key Exchange, with David Pointcheval).
- UMTS and LTE Authentication and Key Agreement (Joe-Kai Tsay and Stig F. Mjølsnes).
- SSH Transport Layer Protocol (with David Cadé).
- Signal (with Nadim Kobeissi and Karthikeyan Bhargavan).
- TLS 1.3 (with Nadim Kobeissi and Karthikeyan Bhargavan).
- ARINC823 public-key and shared-key (avionic protocols).

Extraction of models from implementations

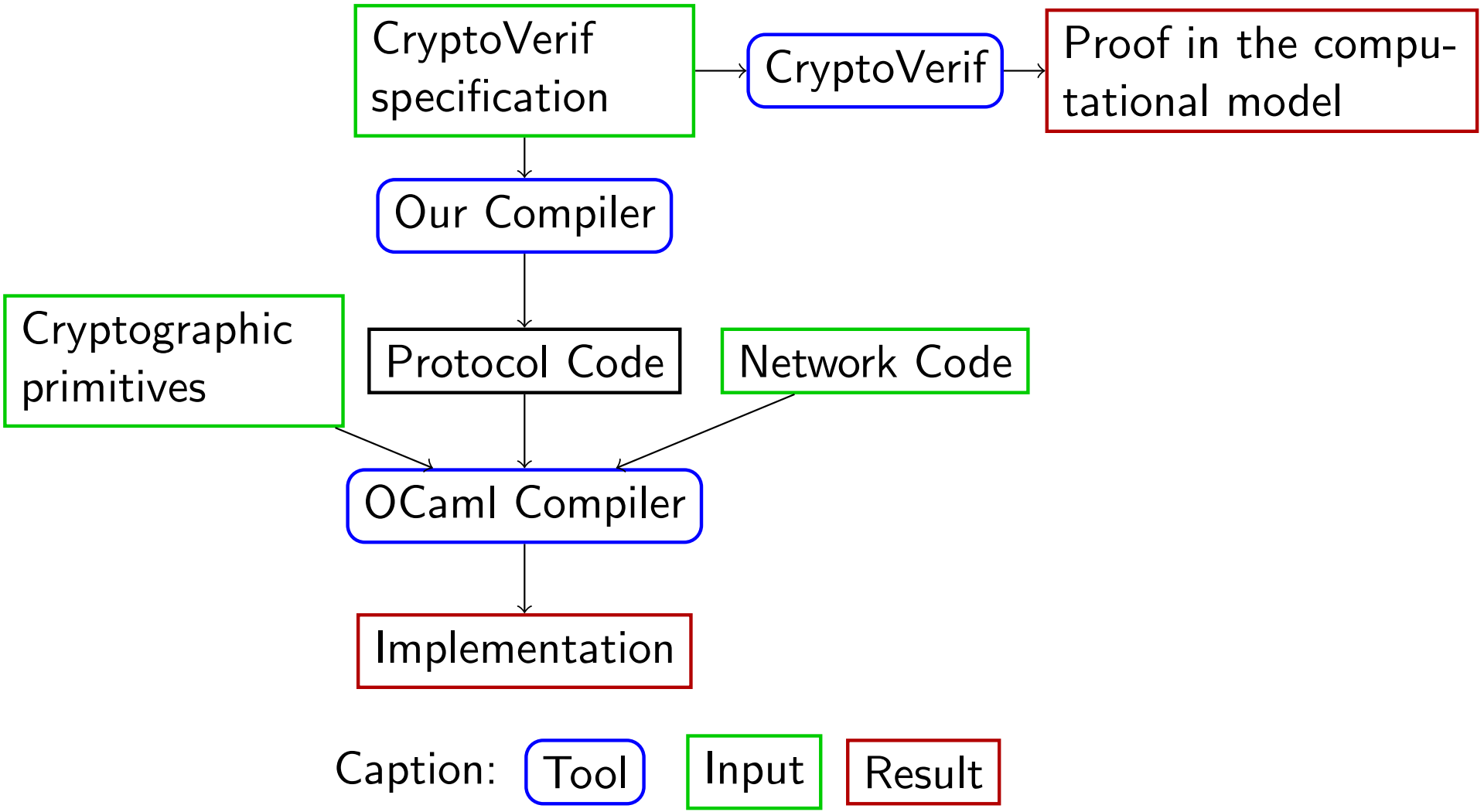
Two approaches:

- A part of an F# implementation of the TLS 1.2 transport protocol (Karthikeyan Bhargavan, Ricardo Corin, Cédric Fournet, and Eugen Zalinescu).
- Extraction from C, for a single trace (Mihhail Aizatulin, Andrew Gordon, and Jan Jürjens).

Difficulties:

- The extracted script must be as simple as possible for CryptoVerif to work.
- CryptoVerif sometimes needs guidance; that may be more difficult with a generated script.

Generation of implementations



Generation of implementations

- Easier to implement than model extraction.
- The script can be tuned to facilitate the proof.
- The proof of soundness of the compiler is complicated.
- Limitations:
  - The language of CryptoVerif is a process calculus.
    It limits what we can express in the implementation.
  - No side-channel protection.

Our SSH implementation interoperates with OpenSSH, but is still a toy implementation.

Demo

Demo
TLS 1.3 Initial handshake

Strengths and limitations (1)

- CryptoVerif can do some reasoning on primitives, for instance:
  - Full domain hash signature (with David Pointcheval)
  - Encryption schemes of Bellare-Rogaway'93 (with David Pointcheval)

  but it is typically better at verifying protocols.
- Proofs of protocols typically include many cases.
  CryptoVerif makes sure that no case is forgotten.
- CryptoVerif cannot perform subtle cryptographic reasoning (e.g. forking lemma).

Strengths and limitations (2)

- CryptoVerif produces proofs.
- It does not reconstruct attacks.
  - When there is an attack, the proof fails.
- Proof failure can come from many reasons:
  - Attack
  - Limitations of CryptoVerif
  - Assumptions too weak
  - Inappropriate or missing guidance
  - ...

  When the proof fails, it is sometimes difficult to understand why and what to do.
- Extensions of CryptoVerif may be required when a proof fails.

Future directions (1)

- Case studies suggest extensions.
- Example: the case study of TLS suggested adding game transformations,
  - for guessing the tested session;
  - for removing parts of games for which can prove that the adversary never wins when it executes them.

  That would allow us to
  - prove some properties under DDH (instead of GDH);
  - prove forward secrecy wrt. the compromise of the pre-shared key.

Future directions (2): reduce the size of games

- The size of games tends to grow:
  - Many cases to distinguish.
  - Complicates guidance since we need to inspect the games.
  - May lead to memory exhaustion.
- Two ideas to improve:
  - Apply cryptographic transformations only to some occurrences of the primitives.
    - to avoid distinguishing cases when it is not really necessary.
  - Merge cases when distinction is not needed until the end of the protocol.
    - New construct: (**if** ... **then** ... **else** ...); ...
- More modularity would also help.

Future directions (3): mutable state

- Support for mutable state is limited.
  - Can handle a counter distinct in each execution.
  - Can model a replay cache.
  - Not much more than that.
- Improve support for mutable state would be helpful:
  - Primitives with internal state.
  - Loops with state in protocols.

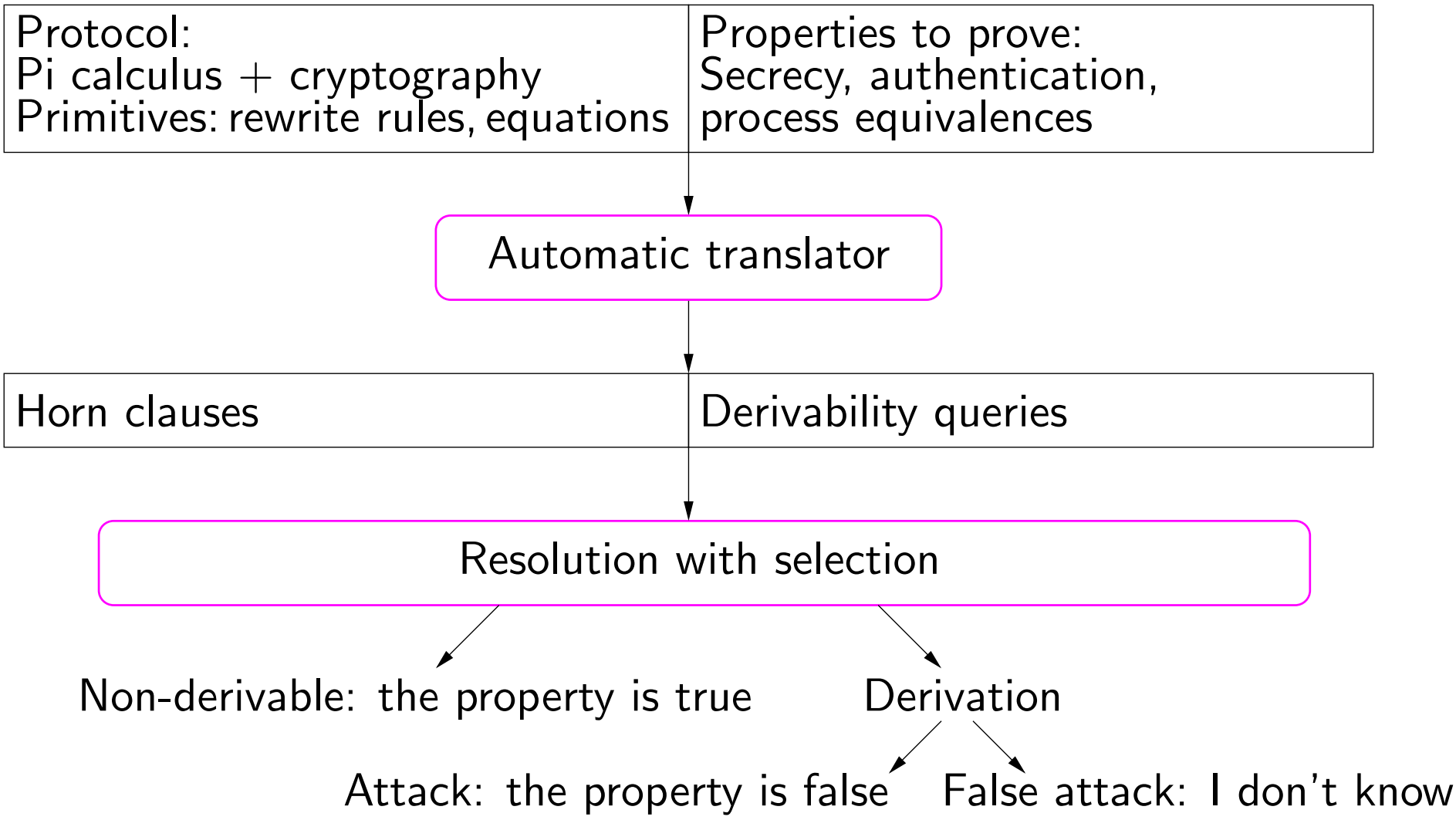
Future directions (4): certification

- The whole CryptoVerif code needs to be trusted.
  - 42000 lines of OCaml
- Produce certificates for proofs
  - Ideally, verifiable by a theorem prover such as Coq
  - Verifying them by another tool such as EasyCrypt would already be a nice step.
- Long-term goal!

ProVerif, http://proverif.inria.fr

- Symbolic model.
- Fully automatic.
- 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, some equivalences.
- Does not always terminate and is not complete. In practice:
  - Efficient: small examples verified in less than 0.1 s; complex ones in a few minutes; very complex ones may take hours.
  - Very precise: no false attack on 19 basic protocols for secrecy and authentication.

ProVerif


Protocol:
Pi calculus + cryptography
Primitives: rewrite rules, equations
Properties to prove:
Secrecy, authentication,
process equivalences
Automatic translator
Horn clauses
Derivability queries
Resolution with selection
Non-derivable: the property is true
Derivation
Attack: the property is false
False attack: I don't know

Complementarity between ProVerif and CryptoVerif

- ProVerif is easier to use.
  - Fully automatic.
- ProVerif finds attacks.
- ProVerif works in the symbolic model.
  - Its proofs are weaker than those of CryptoVerif.
- $\Rightarrow$ use ProVerif in the early design stages;
  confirm security in the computational model using CryptoVerif,
  when ProVerif finds no attack.
- CryptoVerif can prove some protocols that ProVerif cannot!
  - Mutable state: replay cache, counters.
  - ProVerif exhausts memory or takes too much time while CryptoVerif succeeds (e.g. improved ARINC823 public key protocol).

Common input language for ProVerif and CryptoVerif

- The input languages of ProVerif and CryptoVerif are similar.
- Still small differences:
  - Process macros with arguments or not.
  - Syntax for security properties.
- We plan to remove these differences.
  - Same input file for both tools.
  - Still, assumptions on primitives will always be different. They can be put in a library.

EasyCrypt, https://www.easycrypt.info/

- See first talk this morning.
- Less automatic than CryptoVerif.
  - Need to provide the games.
  - To guide the proof that they are indistinguishable.
- Can perform more subtle proofs.
- Better for proving primitives.
- Guidance too difficult for proving complex protocols.

Collaboration between EasyCrypt and CryptoVerif

- Use EasyCrypt to prove properties of primitives.
  Use these properties in CryptoVerif.
- Use CryptoVerif to perform the easy steps of a proof.
  Perform in EasyCrypt the steps that CryptoVerif cannot do.
- Use CryptoVerif as a powerful back-end for EasyCrypt,
  in addition to SMT solvers.
- Challenges: discrepancies
  - Probabilistic list lookup in CryptoVerif
  - Parallelism and sessions explicit in CryptoVerif, not in EasyCrypt

Scary (by Guillaume Scerri)

- Fully automatic
- Sound in the computational model
- Based on the Bana-Comon-Lundh logic
- Preliminary

The model behind Scary (Gergei Bana, Hubert Comon-Lundh)

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, ...

$\Rightarrow$ 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.)

Computational soundness result

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.

Scary implements a decision procedure for this logic.

Cooperation between Scary and CryptoVerif

- Use Scary to find computational attacks.
  - Avoid losing time trying to prove a property that does not hold.
- Use Scary to find the game transformations to apply.
  - The proof can then be reproduced in CryptoVerif.
  - CryptoVerif evaluates the advantage of the adversary. (Scary does not.)

Conclusion

- Each tool has its strengths and weaknesses.
- No tool can do all interesting cryptographic proofs.
- Build a framework of cooperating tools.
- Non-trivial challenge since each tool has its own framework.