Abstracting Cryptographic Protocols by Prolog Rules

Bruno Blanchet

INRIA Rocquencourt
B.P. 105, 78153 Le Chesnay Cedex, France
Bruno.Blanchet@inria.fr

Abstract Most current cryptographic protocol verifiers meet the state space explosion problem, and have to limit the number of executions of the considered protocol during the verification. To solve these problems, we introduce an abstract representation of cryptographic protocols, based on Prolog rules, and use it to verify secrecy properties of protocols.

1 Introduction

This short summary is a reference for the author's invited talk for SAS'01. It presents a new technique for verifying cryptographic protocols, that is described in detail in [5]. The design of cryptographic protocols is difficult and error-prone. This can be illustrated by attacks found in existing protocols [1, 4, 8, 13]. It is therefore important to have automatic tools to verify these protocols. Many different techniques have been used in this area: model-checking [13], rewriting [9], theorem proving [15], typing [2], abstract interpretation [6, 11, 14] (more references can be found in [5]). One of the most widely used techniques is model-checking, but it meets two important problems:

- A first problem is the state space explosion: even for small specifications of protocols, the number of states to explore during verification is very large. Then, the verification of complex protocols becomes impossible.
- Most model-checking tools limit the number of the runs of the protocol, to ensure the termination of the verification process. Indeed, at each run, the protocol creates new values. For an unbounded number of runs, an unbounded number of new values would be needed, and the number of states would be infinite. However, limiting the number of runs of the protocol has serious consequences: an attack that appears with more runs will remain undiscovered. This is a problem for the certification of protocols, and manual proofs have to be done to show that the result obtained for a small number of runs extends to an unbounded number.

These problems have already been tackled by previous works: [17] avoids the state space explosion by using the Strand Space Model, but still sometimes limits the number of runs of the protocol to guarantee termination. [7, 16] do not limit the number of runs of the protocol, by recycling the new values created at each run. However, they limit the number of *parallel* runs of protocols.

We introduce an abstract representation of cryptographic protocols, based on Horn clauses (the basis of Prolog), that solves these problems by performing well-chosen approximations that preserve attacks. This representation yields a more efficient analysis than [14], by representing rules that generate the knowledge of the attacker, instead of representing this knowledge by tree automata. It is also more precise than [6], since our analysis is relational, and it distinguishes values created at the same point of the protocol in different sessions.

2 Representation of cryptographic protocols

We assume that the protocol is executed in the presence of an attacker that can listen to messages, compute, and send messages.

In our representation, messages are represented by terms. For instance, the term $\mathsf{pencrypt}(c_0, \mathsf{pk}(sk))$ represents the encryption of c_0 under the public key $\mathsf{pk}(sk)$, corresponding to the secret key sk. The facts are of the form $\mathsf{attacker}(M)$ meaning that the attacker may have the term M. Our abstract representation of the protocol consists of inference rules of the knowledge of the attacker. For example, the rule:

$$\mathsf{attacker}(\mathsf{pencrypt}(m, \mathsf{pk}(sk))) \wedge \mathsf{attacker}(sk) \rightarrow \mathsf{attacker}(m)$$

means that the attacker can decrypt a message when it has the secret key: if the attacker has the encrypted message $\mathsf{pencrypt}(m, \mathsf{pk}(sk))$ and the secret key sk, it can obtain the cleartext m.

We also represent the actions of the agents involved in the protocol as inference rules of the knowledge of the attacker. Indeed, if the agent A has received the messages $M_1, \dots, M_n$, and sends the message M, we consider that the attacker has relayed the messages $M_1, \dots, M_n$ and intercepts A's reply M. This yields the rule:

$$\mathsf{attacker}(M_1) \wedge \ldots \wedge \mathsf{attacker}(M_n) \rightarrow \mathsf{attacker}(M).$$

If the attacker has the messages $M_1, \dots, M_n$, it can send them to A, simulating the beginning of a protocol run, and obtain the reply M.

Using these principles, we can abstract a cryptographic protocol by a set of Horn clauses. This representation is obviously an abstraction of the multi-set rewriting representation [9]: the number of times a message appears is ignored, to remember only that it has appeared. It is also an abstraction of the linear logic representation of [10]. Moreover, we have built an automatic translator from the applied pi calculus [3] (restricted to certain equational theories) to our representation. Our representation is more abstract than most usual ones:

– The messages are not organized into separate runs. For instance, the ith step of the protocol can be repeated several times when the previous steps

have been executed only once. More generally, a step of the protocol can be repeated any number of times as soon as the previous steps have been executed at least once.
– Moreover, the freshness of nonces is modeled by considering fresh values as functions of the messages previously received by the creator of the value. Therefore, the fresh values are considered as different if and only if the previous messages are different. The same nonces are reused in several sessions of the protocol, when the previous messages are the same. Therefore, we can have a finite space even when considering an infinite number of sessions.

These approximations are keys to an efficient verification of the protocols, and enable us not to limit the number of executions of the considered protocol.

3 Verifying secrecy properties

We use our representation to verify secrecy properties of protocols: if the fact $\mathsf{attacker}(M)$ cannot be derived from the rules representing the protocol, then the adversary cannot have M, and the term M remains secret. This is exactly the kind of information that usual Prolog systems compute. However, they do not terminate on the rules representing a protocol, so we have designed, proved correct, and implemented an efficient algorithm to handle this particular situation. This algorithm is fully detailed in [5].

The experimental results show that many examples of protocols of the literature, including Skeme [12], can be analyzed by our tool with very small resources: from less than 0.1 s for simple protocols to 23 s for the main mode of Skeme, and less than 2 Mb of memory.

References

[1] Martín Abadi. Explicit Communication Revisited: Two New Attacks on Authentication Protocols. *IEEE Transactions on Software Engineering*, 23(3):185–186, March 1997.

[2] Martín Abadi. Secrecy by typing in security protocols. In *Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes on Computer Science*, pages 611–638. Springer Verlag, 1997.

[3] Martín Abadi and Cédric Fournet. Mobile Values, News Names, and Secure Communication. In *28th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, London, United Kingdom, January 2001. ACM Press.

[4] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[5] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, Cape Breton, Canada, June 2001.

[6] Chiara Bodei. *Security Issues in Process Calculi*. PhD thesis, Università di Pisa, January 2000.

[7] Philippa Broadfoot, Gavin Lowe, and Bill Roscoe. Automating Data Independence. In *6th European Symposium on Research in Computer Security (ESORICS 2000)*, volume 1895 of *Lecture Notes on Computer Science*, pages 175–190, Toulouse, France, October 2000. Springer Verlag.

[8] Michael Burrows, Martín Abadi, and Roger Needham. A Logic of Authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, February 1989.

[9] Grit Denker, Jose Meseguer, and Carolyn Talcott. Protocol Specification and Analysis in Maude. In N. Heintze and J. Wing, editors, *Proc. of Workshop on Formal Methods and Security Protocols*, Indianapolis, Indiana, 25 June 1998.

[10] N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols (FMSP'99)*, Trento, Italy, 5 July 1999.

[11] Jean Goubault-Larrecq. A Method for Automatic Cryptographic Protocol Verification (Extended Abstract), invited paper. In *Fifth International Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'2000)*, Cancún, Mexique, May 2000. Springer-Verlag.

[12] Hugo Krawczyk. SKEME: A Versatile Secure Key Exchange Mechanism for Internet. In *Proceedings of the Internet Society Symposium on Network and Distributed Systems Security*, February 1996. Available at `http://bilbo.isu.edu/sndss/sndss96.html`.

[13] Gavin Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes on Computer Science*, pages 147–166. Springer Verlag, 1996.

[14] David Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *Static Analysis Symposium (SAS'99)*, volume 1694 of *Lecture Notes on Computer Science*, pages 149–163. Springer Verlag, September 1999.

[15] Larry C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.

[16] A. W. Roscoe and P. J. Broadfoot. Proving Security Protocols with Model Checkers by Data Independence Techniques. *Journal of Computer Security*, 7(2, 3):147–190, 1999.

[17] Dawn Xiaodong Song. Athena: a New Efficient Automatic Checker for Security Protocol Analysis. In *Proc. of 12th IEEE Computer Security Foundation Workshop (CSFW-12)*, Mordano, Italy, June 1999.