Automatic Verification of Cryptographic Protocols: A Logic Programming Approach

Bruno Blanchet
CNRS, Département d'Informatique, École Normale Supérieure, Paris
and Max-Planck-Institut für Informatik, Saarbrücken
Bruno.Blanchet@ens.fr

ABSTRACT

We present a technique for cryptographic protocol verification, based on an intermediate representation of the protocol by a set of Horn clauses (a logic program). This technique makes it possible to verify security properties of the protocols, such as secrecy and authenticity, in a fully automatic way. Furthermore, the obtained security proofs are valid for an unbounded number of sessions of the protocol.

Categories and Subject Descriptors

D.1.6 [**Programming Techniques**]: Logic Programming; D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods, Validation*; D.4.6 [**Operating Systems**]: Security and Protection—*Cryptographic controls*

General Terms

Security, Verification.

Keywords

Cryptographic protocols, Logic programming, Horn clauses, Automatic verification.

1. INTRODUCTION

The goal of security protocols is to exchange sensitive data (for instance credit card numbers in e-commerce applications) on an insecure network such as Internet. Data are protected from hackers on the network by using cryptographic primitives such as encryption. The design of cryptographic protocols is particularly difficult and error-prone. An extreme example is the well-known Needham-Schroeder public key protocol [13]: In this very simple protocol, a flaw was found 17 years after its publication, by Lowe [12] using the model checker FDR. This surprising complexity in an apparently simple specification mainly comes from the fact that errors appear only in the presence of malicious users that try to break the protocol. Errors then remain undetected during testing, and manual verification also often misses potential attacks. That is a reason why the verification of cryptographic protocols has received a lot of interest in computer science research, and is probably still receiving more and more. It is indeed an area of choice for the application of formal methods, which can provide formal proofs of the security properties of cryptographic protocols.

Most works on the automatic verification of cryptographic protocols, including ours, are done in the so-called Dolev-Yao model [9]. The protocol is assumed to be executed in the presence of an adversary that can intercept all messages, decompose them, recompose its own messages, and send them as if they came from a honest participant. Moreover, cryptographic primitives such as encryption are assumed to be perfect: one can decrypt only if one has the decryption key.

In our presentation, we focus on a verification technique based on logic programming, described below. This technique has the following characteristics:

- It is fully automatic. The user gives only the specification of the protocol and the properties to verify.
- It can handle a wide range of cryptographic primitives, including shared- and public-key cryptography (encryption and signatures), hash functions, and even a simple model of Diffie-Hellman key agreements.
- In contrast to pure finite state techniques, it can verify protocols without arbitrarily bounding the number of executed sessions (even in parallel) of the protocol or the size of messages. This makes it possible to obtain actual proofs of the security properties.
- Up to now, we verified secrecy and authenticity properties using this method.
- It is efficient in practice. Many examples from the literature can be verified in less than 0.1 s, and it takes a few minutes to verify complex practical examples.

In this short abstract, we are not going to compare our method with the many other techniques for cryptographic protocol verification. We refer the reader to [1, 6] for a comparison with related work.

2. OVERVIEW OF THE METHOD

The verification method is summarized in Figure 1. The Horn clause verification technique is not specific to any formalism for representing the protocol. Among the many existing formalisms, we focused on extensions of the pi calculus

PPDP'03, August 27–29, 2003, Uppsala, Sweden

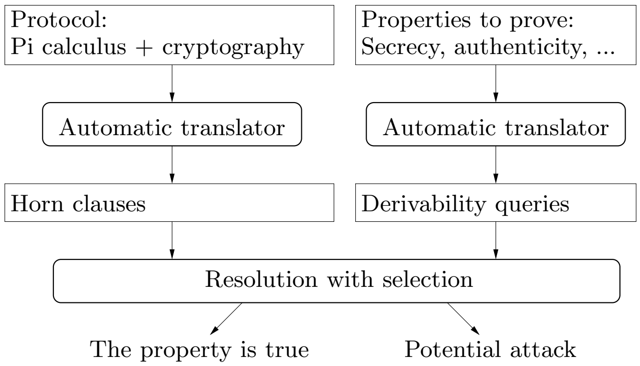

Figure 1: The verification process

with cryptographic primitives. These process calculi provide minimal programming languages that are particularly well-suited for specifying cryptographic protocols. This line of research has been pioneered by the spi calculus [4], which adds encryption, signatures, and hash functions to the pi calculus. It has been considerably extended by the applied pi calculus [3], which provides a generic treatment of cryptographic primitives, defined by an equational theory. In our work, we focus on a simpler case in which cryptographic primitives are defined by reduction relations. This case can still represent many cryptographic primitives. We distinguish two kinds of primitives: constructors and destructors. Constructors, such as encryption $\mathsf{encrypt}$, build new terms, while destructors, such as decryption $\mathsf{decrypt}$, manipulate terms. Destructors are defined by reduction relations. For example, the shared-key decryption can be defined by the following reduction relation:

$$\mathsf{decrypt}(\mathsf{encrypt}(x, y), y) \to x$$

Decrypting a ciphertext $\mathsf{encrypt}(x, y)$ with the same key as the encryption key y, yields the cleartext x.

The protocol represented in this calculus is then automatically translated into a set of Horn clauses (a logic program). This translation is defined in [1]. The main idea of the Horn clause representation is to use a predicate $\mathsf{attacker}$, such that $\mathsf{attacker}(M)$ means "the attacker may have the message M". For example, the fact that the attacker can encrypt, resp. decrypt, when it has the key is represented by the following two clauses:

$$\mathsf{attacker}(x) \wedge \mathsf{attacker}(y) \to \mathsf{attacker}(\mathsf{encrypt}(x, y))$$
$$\mathsf{attacker}(\mathsf{encrypt}(x, y)) \wedge \mathsf{attacker}(y) \to \mathsf{attacker}(x)$$

When the attacker has the cleartext x and the key y, it can built the ciphertext $\mathsf{encrypt}(x, y)$, and when the attacker has the ciphertext and the key, it can obtain the cleartext. The messages exchanged by the honest participants of the protocol can also be represented by similar clauses. They are considered as oracles that the attacker can use to increase its knowledge. When a participant A sends a message M after receiving messages $M_1, \ldots, M_n$, we have a clause:

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

Indeed, when the attacker has $M_1, \ldots, M_n$, it can send it to A; A replies with M, that the attacker can intercept. More details on this representation as well as the coding of an example can be found in [7].

This representation of protocols is approximate in that the application of Horn clauses can be repeated any number of times, while the real protocol repeats each step only once per session. So, the state of the participants is only partly modeled. A model that does not make such an approximation can be obtained by using clauses in linear logic instead of classical logic, to control the number of repetitions of each step [10]. Hence, our technique is sound (when it says that a security property is true, then it is actually so), but not complete (false attacks can be found). However, in our tests, false attacks never occurred. In fact, false attacks would occur only for protocols that first need to keep some data secret, then publish it later in the protocol. In that situation, the Horn clause model considers that the attacker can re-inject the secret in the early part of the run, which is not possible in reality (V. Cortier, personal communication). This approximation is a key to verify protocols without bounding the number of sessions.

Using this representation, secrecy can be inferred from non-derivability: if $\mathsf{attacker}(M)$ is not derivable, then the attacker cannot have M, that is, M is secret. Even if the derivability is undecidable in general, several techniques can be used to determine derivability of a fact from a set of clauses. However, the simplest techniques, such as SLD-resolution, would never terminate (for example the clause for decryption given above immediately leads to a loop). More elaborate resolution techniques succeed in this task: Ordered resolution with selection has been used in [14]. Ordered resolution with factorization and splitting has been shown to terminate on protocols that *blindly copy* at most one message at each step [8]. A blind copy is a protocol step in which a participant creates a new message by reusing a part of a received message without looking at what is inside this part. This class of protocols results in clauses with at most one variable. We have shown with A. Podelski that resolution with free selection (without ordering) terminates on *tagged protocols* [7]. In these protocols, each application of a cryptographic primitive is distinguished from others by a constant (the tag). For example, we use $\mathsf{encrypt}((c_0, m), k)$ for encrypting m under k, instead of $\mathsf{encrypt}(m, k)$. It is easy to add tags, and it is also a good protocol design: it can make the protocol more secure. When we verify a tagged protocol, the implemented protocol should of course also be tagged, since the security proof for the tagged protocol does not imply the security of a non-tagged version. A key to obtain termination is to avoid resolving on facts of the form $\mathsf{attacker}(x)$. Indeed, these facts resolve with all facts of the form $\mathsf{attacker}(M)$, which leads to non-termination in almost all examples coming from protocols. The last three techniques terminate on numerous practical examples, even outside the decision classes mentioned above.

In case $\mathsf{attacker}(M)$ is derivable from the representation of the protocol, the system cannot prove secrecy. The derivation of $\mathsf{attacker}(M)$ can then be used to reconstruct an attack. (Such a reconstruction can fail if the system has found a false attack.) We have not yet automated this reconstruction, although we believe that this would be feasible.

The technique can be extended to other security properties, such as authenticity [6]. Intuitively, authenticity means that if a participant A believes that she has executed a session of the protocol with B, then B has started a session with A. Moreover, A and B agree on certain parameters of the protocol, such as session keys. Following Woo and Lam [15], authenticity is formalized by *correspondence assertions* of the form: if A executes an event $\mathsf{end}(M)$ (A says she has executed a session with B), then B must have executed an event $\mathsf{begin}(M)$ (B says he has started a session with A). We can also require that the number of executions of the begin event is at least the number of executions of the end event. We prove such correspondence assertions and more general ones involving several begin events [2].

Using our tool, we verified numerous protocols from the literature, finding known attacks or proving the correctness of the protocols. Most examples were verified in less than 0.1 s [6]. In collaboration with M. Abadi, we also applied our technique to the verification of a recent certified email protocol [2]. This protocol uses a secure channel between two participants. In one version that we verified, this channel is established by using the SSH (secure shell) transport layer protocol. This protocol was verified in 42 s on an Intel Xeon 1.7 GHz. Our cryptographic protocol verifier can be freely downloaded at

`http://www.di.ens.fr/~blanchet/crypto-eng.html`

3. FUTURE WORK

Even if much work has already been done on the verification of cryptographic protocols, there is still much to do. Some features of protocols, such as timestamps, are not represented in our model. Moreover, we believe that much progress could still be done on the verification of more subtle properties such as denial of service or observational equivalence of processes. (Intuitively, two processes are observationally equivalent when the attacker cannot distinguish them. Observational equivalence is a powerful means of specifying security properties.) At last, we assume perfect cryptography. A more satisfactory model would use the more realistic computational view of cryptography. Even if some important work has already been done on the link between the formal and computational views of cryptography (see for example [5]) and on the analysis of protocols in the computational setting [11], the current state of the art is still far from a fully automatic verification of protocols in that setting.

4. ACKNOWLEDGMENTS

We would like to thank Martín Abadi for being at the origin of our work on cryptographic protocols and for suggesting improvements in Figure 1.

5. REFERENCES

[1] M. Abadi and B. Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. In *29th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL 2002)*, pages 33–44. ACM, Jan. 2002.

[2] M. Abadi and B. Blanchet. Computer-Assisted Verification of a Protocol for Certified Email. In R. Cousot, editor, *Static Analysis, 10th International Symposium (SAS'03)*, volume 2694 of *LNCS*, pages 316–335. Springer, June 2003.

[3] M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In *28th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM, Jan. 2001.

[4] M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, Jan. 1999. An extended version appeared as Digital Equipment Corporation Systems Research Center report No. 149, January 1998.

[5] M. Abadi and P. Rogaway. Reconciling two views of cryptography (The computational soundness of formal encryption). In *Proceedings of the First IFIP International Conference on Theoretical Computer Science*, volume 1872 of *LNCS*, pages 3–22. Springer, Aug. 2000.

[6] B. Blanchet. From Secrecy to Authenticity in Security Protocols. In M. Hermenegildo and G. Puebla, editors, *9th International Static Analysis Symposium (SAS'02)*, volume 2477 of *LNCS*, pages 342–359. Springer, Sept. 2002.

[7] B. Blanchet and A. Podelski. Verification of Cryptographic Protocols: Tagging Enforces Termination. In A. Gordon, editor, *Foundations of Software Science and Computation Structures (FoSSaCS'03)*, volume 2620 of *LNCS*, pages 136–152. Springer, Apr. 2003.

[8] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In R. Nieuwenhuis, editor, *In Proc. 14th Int. Conf. Rewriting Techniques and Applications (RTA'2003)*, volume 2706 of *LNCS*. Springer, June 2003.

[9] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, Mar. 1983.

[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)*, 5 July 1999.

[11] P. Laud. Handling Encryption in an Analysis for Secure Information Flow. In P. Degano, editor, *Programming Languages and Systems, 12th European Symposium on Programming, ESOP'03*, volume 2618 of *LNCS*, pages 159–173. Springer, Apr. 2003.

[12] G. 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 *LNCS*, pages 147–166. Springer, 1996.

[13] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM*, 21(12):993–999, Dec. 1978.

[14] C. Weidenbach. Towards an Automatic Analysis of Security Protocols in First-Order Logic. In H. Ganzinger, editor, *16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *LNAI*, pages 314–328. Springer, July 1999.

[15] T. Y. C. Woo and S. S. Lam. A Semantic Model for Authentication Protocols. In *Proceedings IEEE Symposium on Research in Security and Privacy*, pages 178–194, May 1993.