Safe abstractions of data encodings in formal security protocol models

Alfredo Pironti\textsuperscript{1} and Riccardo Sisto\textsuperscript{2}

\textsuperscript{1} Prosecco, INRIA Paris-Rocquencourt, 23, Avenue d'Italie, 75013 Paris, France
\textsuperscript{2} Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi, 24, 10129 Torino, Italy

Abstract. When using formal methods, security protocols are usually modeled at a high level of abstraction. In particular, data encoding and decoding transformations are often abstracted away. However, if no assumptions at all are made on the behavior of such transformations, they could trivially lead to security faults, for example leaking secrets or breaking freshness by collapsing nonces into constants.

In order to address this issue, this paper formally states sufficient conditions, checkable on sequential code, such that if an abstract protocol model is secure under a Dolev–Yao adversary, then a refined model, which takes into account a wide class of possible implementations of the encoding/decoding operations, is implied to be secure too under the same adversary model. The paper also indicates possible exploitations of this result in the context of methods based on formal model extraction from implementation code and of methods based on automated code generation from formally verified models.

Keywords: Model abstraction, Refinement, Security protocols

1. Introduction

In the last years, several techniques based on formal methods have been developed to analyze abstract models of security protocols. These models, initially introduced by Dolev and Yao \cite{DY83}, represent messages as instances of high level abstract data types. This high abstraction level makes automated verification viable, so that Dolev–Yao verification is now a well-established technique for security protocol verification, even within reach of non-experts. However, one question arises about how to ensure that the logical correctness of an abstract protocol is preserved when more concrete versions of the protocol are defined and when their implementations are developed using programming languages.

This paper focuses on one research line that consists of extending the application of the Dolev–Yao approach from very abstract models, where only the main message components and working scenarios are considered, to more detailed models capturing more closely all the real data structuring and the real program flow of protocol implementations.

Correspondence and offprint requests to: R. Sisto, E-mail: riccardo.sisto@polito.it
Two different strategies have been proposed in order to cope with this extension: automatic code generation from abstract models, e.g. [PSD04, TH04, GHR05], and automatic model extraction from implementation code, e.g. [BFGT06, Jür05, GLP05, BFG06].

Methods based on automatic code generation start from a high-level, formally verified, specification of the protocol, which abstracts away from many details about cryptographic and communication operations and data representations, and fill the semantic gap between formal specification and implementation, guided by implementation choices provided by the user. In [PS10], a formally sound algorithm is provided to automatically translate abstract models to source Java code. Notably, the code responsible for data transformations is not automatically generated, potentially allowing security flaws to be introduced by incorrect manual implementation of such code. Indeed, in the case study reported in [PPS12], about 30% of the code is dealing with data transformations and is manually implemented.

Methods based on automatic model extraction start from an already existing, full blown implementation code, from which an abstract model is extracted and formally verified. In this case, a formal soundness proof has been given for the method presented in [BFGT06]. One of the things that can be observed by looking at the results reported in [BFGT06, Jür05, BFG06], is that the part of the extracted formal model that describes data encoding and decoding operations can be quite complex, as big in size as the rest of the protocol model. This occurs even though in [BFGT06, BFG06] the implementations of some low-level library operations, such as those for basic XML manipulation, are not included in the model but rather assumed to correctly refine their symbolic counterpart.

The wrong implementation of data transformations may be responsible for security faults. For this reason, it is not possible to simply neglect them when analyzing security protocols. For example, consider this very simple RPC-like protocol in the Dolev–Yao model (where perfect encryption with a private shared key also subsumes authentication), expressed abstractly in Alice and Bob notation:

1: A → B : (n, M, REQ)_{K_{A,B}}; where n is a nonce and REQ a constant tag

2: B → A : (n, f(M), RES)_{K_{A,B}}; where RES is a constant tag

Assume that, before sending message 2, B emits a begin(A, B, n) event, meaning that a session of the protocol was started between A and B with nonce n, and that, when receiving message 2, A first checks that the received tag is RES and the received n matches the local one, and only then emits an end(A, B, n) event, meaning that a session between A and B with nonce n was correctly terminated. On this abstract model, assuming Kab is initially not known by the adversary, one can prove the injective correspondence end(A, B, n) = begin(A, B, n), meaning that, even in the presence of a Dolev–Yao adversary, in each execution of the protocol each end(A, B, n) has its own corresponding begin(A, B, n).

Now consider a refined model, where each field of the encrypted content of a message is encoded before applying encryption. Suppose the correct encoding for REQ is 0 and the correct encoding for RES is 1, but the implementations of the encoding and decoding transformations used by A and B have some bugs. More precisely, suppose that, erroneously, e_A(REQ) = 1, where e_A(·) is the implementation of the encoding transformation used by A. Suppose also that B uses a different implementation having the reverse bug, i.e. d_B(1) = REQ, where d_B(·) is the implementation of the decoding transformation used by B. Because of these errors, both message 1 and 2 have the same value for the tag and the refined protocol model has a security flaw, because the adversary can play message 1 back to A, and A will accept her own message as a valid message 2, breaking the injective agreement. In conclusion, formal analysis can catch this flaw if using a detailed model, close to the real implementation, while the flaw is missed if using a more abstract model.

This kind of errors does not necessarily affect interoperability (in the previous example, A and B can run the protocol successfully despite their errors). This implies more difficulty in discovering such errors by classical program testing.

The aim of this paper is to formally state and prove sufficient conditions under which the detailed models of data transformations, such as the ones extracted from protocol code in [BFG06], can be avoided and replaced by much simpler models or assumptions that can be checked on sequential code and in isolation (i.e. without considering the behavior of the adversary), while obtaining the same kind of security assurance on the protocol implementation.