Computational Secrecy by Typing for the Pi Calculus

Martín Abadi[1,2], Ricardo Corin[1,3], and Cédric Fournet[1]

[1] Microsoft Research
[2] University of California, Santa Cruz
[3] University of Twente

Abstract. We define and study a distributed cryptographic implementation for an asynchronous pi calculus. At the source level, we adapt simple type systems designed for establishing formal secrecy properties. We show that those secrecy properties have counterparts in the implementation, not formally but at the level of bitstrings, and with respect to probabilistic polynomial-time active adversaries. We rely on compilation to a typed intermediate language with a fixed scheduling strategy. While we exploit interesting, previous theorems for that intermediate language, our result appears to be the first computational soundness theorem for a standard process calculus with mobile channels.

1 Introduction

In security, both attacks and defenses can operate at various levels of abstraction. For a distributed program, reasoning about security can be in terms of programming-language constructs and concepts, or in terms of their implementations. When those implementations use cryptography, the cryptographic primitives may be represented as black boxes, as specific functions on bitstrings, or even as computing processes with timing and power-consumption characteristics that an attacker may attempt to exploit. While programming abstractions for security can be helpful, they should ideally be mapped to concrete implementations that resist realistic low-level attacks.

In the last decade, a substantial research effort has started to address this problem (e.g., [1, 5, 7, 9, 11–13, 17, 19]). In this paper, we contribute to this line of work by investigating an implementation of a concurrent language with message passing and channel mobility. We treat cryptography both formally (in terms of symbolic expressions) and computationally (at the level of bitstrings, with resource-bounded adversaries).

Specifically, we define and study a distributed cryptographic implementation for an asynchronous pi calculus. At the source level, we adapt simple type systems designed for establishing formal secrecy properties. In particular, we rely on secrecy types for asymmetric communication, in the style of the local pi calculus [3, 18], and on the name-confinement guarantees implied by putting names into scoped groups [14]. We show that those secrecy properties have strong computational counterparts in the implementation, with respect to probabilistic polynomial-time active adversaries that operate on concrete bitstrings.

The implementation leverages Laud's recent results [17] on secrecy by typing in the context of a simulatable cryptographic library [9, 11, 12]. Laud has defined a restricted variant of the spi calculus [6] with a fixed scheduling strategy and without channel mobility (so with fixed, global communication ports). We use Laud's calculus as an

N. Kobayashi (Ed.): APLAS 2006, LNCS 4279, pp. 253–269, 2006.

intermediate language: we translate the pi calculus to his calculus, then rely on his use of the simulatable cryptographic library. Laud employs a type system for secrecy and proves its soundness with respect to the cryptographic library. We show that our translation is type-preserving. Then, via Laud's results, we obtain computational secrecy guarantees, as a soundness theorem for our pi calculus typings.

Related Work. The comparison of formal and computational cryptography is an active research field (e.g., [7, 11, 17, 19]); it has produced computational justifications for formal models of cryptographic operations and for classes of protocols that use formal cryptography. At a higher level, we have implementations of process calculi in terms of black-box, formal cryptography (e.g., [1, 4, 5]). It might be tempting to try to compose the results from those two efforts. For instance, one might imagine a translation from the pi calculus to Turing machines via the spi calculus. Unfortunately, this strategy is not viable at present, and may never be. First, compiling the pi calculus to the spi calculus while preserving security guarantees is difficult at best [1]. In addition, we lack a full computational interpretation for the pi or the spi calculus; in particular, the pi calculus features non-determinism and non-termination, which seem at odds with probabilistic polynomial-time computation. Type systems do help, as does a certain realism in setting goals—for instance, aiming to preserve only secrecy properties, and not necessarily all testing equivalences. Alternatively, one may alter the pi calculus to reflect implementation constraints; Adão and Fournet [8] thus designed a calculus with mobile names (but not mobile channels) and ad hoc communications primitives, and established the computational soundness of its implementation for observational equivalence. Other works also develop implementations of abstract security functions. In particular, Canetti and Krawczyk have considered the problem of implementing secure channels [13], without however a language framework.

Our main result appears to be the first computational soundness theorem for a standard process calculus with mobile channels. In fact, the literature does not seem to contain even a computational soundness theorem for CCS. Going beyond CCS, the main difficulties that we address pertain to channel scopes and mobility, which are central to the pi calculus. Secrecy by typing can be regarded as a discipline for that mobility.

Contents. Section 2 defines our source language. Section 3 presents a local type system. Section 4 explains the intermediate language. Section 5 describes a distributed implementation of the asynchronous pi calculus. Section 6 presents the computational secrecy result. Section 7 considers the addition of name groups. Section 8 concludes.

2 The Source Language

This section introduces our source process calculus, by giving its syntax and semantics. It also discusses secrecy, informally.

The syntax of the calculus appears in Figure 1. It assumes an infinite set of names and an infinite set of variables; a, b, c, k, s, and similar identifiers range over names, and x, y, and z range over variables. The syntax distinguishes a category of terms (data) and processes (programs). The terms are variables and names. The processes include constructs for communication, concurrency, and dynamic name creation, roughly those of the pi calculus, and a conditional. The calculus is polyadic, in the sense that messages