Language Design for Computationally Sound Communications Abstractions

Pedro Adão[1]* and Cédric Fournet[2]

[1] Center for Logic and Computation, IST, Lisboa, Portugal
[2] Microsoft Research

Abstract. We are interested in computationally sound implementations for languages of distributed communicating processes, with secure high-level primitives for authentication and secrecy, but without explicit cryptography. We develop such an implementation for a variant of the pi calculus [6]. In this language, security properties can be studied using traces and equivalences that account for the presence of an arbitrary (abstract) adversary that controls the network. The cryptographic implementation of the language uses standard primitives and assumptions; it guarantees that these abstract properties also hold in a concrete, distributed setting, against probabilistic polynomial-time active adversaries.

At FCC'06, we intend to review and discuss the design space for programming languages with cryptographically sound implementations. In particular, we motivate some unusual design choices for our language, we discuss its current limitations, and we consider possible extensions.

1 Computational Soundness for Communicating Processes

When designing and verifying secure systems, a certain level of idealization is needed to provide manageable mathematical treatment. Accordingly, two views of cryptography have evolved over the years. In the first view, cryptographic protocols are expressed algebraically, within simple languages. This formal view is suitable for automated computer tools, but may be too abstract. In the second view, cryptographic primitives are probabilistic algorithms that operate on bitstrings. This view involves probabilities and limits in computing power; it is harder to handle formally, especially when dealing with large protocols. Getting the best of both views is appealing, and is the subject of active research that aims at building security abstractions with formal semantics and sound computational implementations. Many recent works focus on the soundness of cryptographic primitives specified using an abstract, Dolev-Yao semantics [4, 5, 7–9, 11–13]. Although their results yield stronger cryptographic guarantees, they do not suffice in general to establish the security properties of programs that use cryptographic protocols.

* Partially supported by FCT grant SFRH/BD/8148/2002. Additional support from FEDER/FCT project Fiblog POCTI/2001/MAT/37239 and FEDER/FCT project QuantLog POCI/MAT/55796/2004.

To this end, we are interested in the design and implementation of distributed process calculi with higher-level security primitives. Hence, at ICALP'06, we present a first computationally sound and complete implementation for a variant of the pi calculus. In this position paper, we only outline our approach—we refer to [6] for additional definitions, results, discussions, and related work.

The calculus features distributed principals, reliable messaging, name mobility, and authentication primitives, but neither explicit cryptography nor probabilistic behaviors. Taking advantage of existing techniques from concurrency, it supports simple reasoning, based on labeled transitions and observational equivalence. We precisely define its concrete implementation. We establish general soundness and completeness results in the presence of active probabilistic polynomial-time (PPT) adversaries, for both trace properties and observational equivalences, essentially showing that high level reasoning accounts for all low-level adversaries.

The concrete implementation relies on standard cryptographic primitives, computational security definitions, and networking assumptions. It also combines typical distributed implementation mechanisms (abstract machines, marshaling and unmarshaling, multiplexing, and basic communications protocol). This puts interesting design constraints on our high-level semantics, as we need to faithfully reflect low-level properties and limitations and, at the same time, remain as abstract as possible. In particular, our high-level environments should be given precisely the same capabilities as low-level PPT adversaries.

Instead, we could have proceeded in two steps, by first compiling high-level communications to an intermediate calculus with formal, explicit cryptography (in the spirit of [2, 3]), then establishing the soundness of this calculus with regards to computational cryptography. However, this second soundness problem seems considerably more delicate than ours, inasmuch as we would need to provide an implementation for a richer calculus that enables any direct usage of formal cryptography. In contrast, for instance, our language mentions principals, but never mentions their underlying cryptographic keys, so no high-level program may ever leak a key, or create an encryption cycle.

2 Language Design Issues

At FCC'06, we intend to discuss our approach, in particular as regards design choices, existing limitations, and possible extensions. We briefly present some of these issues.

Computational Setting There is a discrepancy between implicit, dynamic concurrency in process calculi—reflected in their labeled transition semantics, and even in their syntax for parallel composition—and interactions between concrete Turing machines—traditionally used by cryptographers to define computational indistinguishability. Machines are sequential: one machine runs at a time, and may block the whole computation; moreover, interactions between machines are quite static and limited. For instance, as in Laud's and Blanchet's low-level calculi [10, 12], the machines that implement our processes input one low-level bitstring, run for a bounded amount of time, then output one low-level bitstring.

We do not impose such a discipline on high-level programs, inasmuch as we can safely compile their internal concurrency using local scheduling and multiplexing techniques, and rely on the adversary for global scheduling. This approach seems preferable for establishing abstract security properties, but also requires some care in the implementation, e.g. to prevent information leaks by traffic analysis once the scheduling is fixed.

Communications Primitives As expected in a process calculus, our language supports abstract reliable messaging between principals: message senders and receivers are authenticated; message contents is protected; and messages are delivered at most once to their intended recipient. These guarantees can be systematically enforced using signatures, MACs, encryptions, and anti-replay caches.

On the other hand, we let the adversary in full control of the network: it can intercept, delay, or even block permanently any communication between principals. Hence, the simple pi calculus rule $\overline{c}\langle M\rangle.P \mid c(x).Q \rightarrow P \mid Q\{M/x\}$, which models silent communication "in the ether", would be too abstract for situations where the two processes communicating on channel c are located on different machines. Instead, we use non-standard rules, such that every communication is mediated by the adversary, using one intercepted-output step followed later (if the adversary decides so) by a forwarded-message-input step that carries the same message.

Non-Determinism and Infinite Behaviors The semantics of process calculi naturally feature unbounded, infinite, and non-deterministic computations, which are often crucial for reasoning about process equivalences. On the other hand, low-level implementations are necessarily polynomial and probabilistic. Our approach is to keep an abstract process-calculus semantics, to provide a *partial* polynomial, deterministic implementation for each local process, and to add sufficient side conditions that exclude any behaviors that may not be observable by polynomial adversaries. Although these conditions are serious in principle, they are easily met by our examples and applications so far, and they can (probably) be generalized.

Thus, we exclude any observable internal non-determinism in source processes, the reason being that, in our cryptographic setting, a process could otherwise be used by the adversary as an oracle to guess a particular key in linear time, one bit at a time. However, we still allow external non-determinism, under the control of the adversary. Besides, we require that the size of any high-level trace, measured abstractly, be bounded by a polynomial in the size of the inputs. This excludes, for instance, any brute-force attack by coding a decryption algorithm as a high-level process, and any observation of the cost of an internal computation. Pragmatically, we also provide replicated inputs and pattern matching to avoid trivial sources of divergence and non-determinism.

Wire-Level Traffic Analysis Our adversary observes all traffic, but it may not learn much about its payload. Technically, we distinguish two kinds of output transitions from processes to the environment (which represents the adversary). Intended outputs carry messages sent to a principal controlled by the adversary; their label describe their high-level content. Intercepted outputs carry messages between the observed principals; their content is opaque. For instance,

- if the principal c is controlled by the adversary, the output label $a{:}c\langle \mathtt{Hello}, 3\rangle$ informs the adversary that principal a sends the pair $\mathtt{Hello}, 3$;
- in contrast, if neither a nor c has been compromised, the output label is blinded, of the form $\nu i.a{:}c\langle \_\rangle$ where i represents an opaque handle to the actual message content: the payload identifier i may only be used to forward the message to c.

Our high-level labeled semantics is thus parameterized by an erasure function from explicit labels to blinded labels. This function reflects the amount of information gained by observing traffic; it may be tuned, for example, to further erase the sender or the receiver principals (reflecting some local, anonymous broadcast medium, as in [1]) or on the contrary leak partial information on the message payload (such as its size).

Signature Values and Covert Channels Our language features transferable authentication, in the form of certificates issued by principals. Intuitively, certificates are supported by underlying public-key signatures, but the correspondence between the two turns out to be problematic.

In a first implementation, we attempted to rely solely on the usual computational assumptions on signatures; in order to obtain both soundness and completeness, we had to complicate the high-level semantics of certificates to accommodate the facts that signatures of the same certificate by the same principal may actually be different, and that the adversary may also produce different signature values from those received in certificates. For instance, an adversary could send the same certificate with different signature values to principals a and b, then forward all messages from a and b to c, and finally observe which signature values c uses when sending messages to compromised principals. Hence, the adversary may learn whether c obtained the certificate from a or b. Although this attack is minor, it still needs to be reflected in the high-level semantics to obtain general theorems. Alternatively, our implementation can further require that the signature algorithm be deterministic, thereby shielding the programmer from these technicalities, at the cost of an additional cryptographic assumption.

References

1. M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004. Special issue on Foundations of Wide Area Network Computing. Parts of this work were presented at PET'02 (LNCS 2482) and ISSS'02 (LNCS 2602).
2. M. Abadi, C. Fournet, and G. Gonthier. Authentication primitives and their compilation. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages (POPL 2000)*, pages 302–315. ACM, 2000.
3. M. Abadi, C. Fournet, and G. Gonthier. Secure implementation of channel abstractions. *Information and Computation*, 174(1):37–83, 2002.
4. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
5. P. Adão, G. Bana, and A. Scedrov. Computational and information-theoretic soundness and completeness of formal encryption. In *Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW)*, pages 170–184. IEEE Computer Society Press, 2005.
6. P. Adão and C. Fournet. Cryptographically sound implementations for communicating processes (extended abstract). In *33rd International Colloquium on Automata, Languages and*

Programming (ICALP), LNCS 4052, pages 83–94. Springer-Verlag, 2006. Draft technical report available from `http://research.microsoft/com/~fournet`.

7. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218. IEEE Computer Society Press, 2004.
8. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, pages 220–230. ACM Press, 2003.
9. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. *International Journal of Information Security*, 4(3):135–154, 2005.
10. B. Blanchet. A computationally sound mechanized prover for security protocols. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society Press, 2006. To Appear.
11. V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proceedings of the 14th European Symposium on Programming (ESOP)*, volume 3444 of *LNCS*, pages 157–171. Springer-Verlag, 2005.
12. P. Laud. Secrecy types for a simulatable cryptographic library. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS)*, pages 26–35. ACM Press, 2005.
13. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proceedings of the 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *LNCS*, pages 133–151. Springer-Verlag, 2004.