Cryptographically Sound Implementations for Communicating Processes (Extended Abstract)

Pedro Adão[1]* and Cédric Fournet[2]

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

Abstract. We design a core language of principals running distributed programs over a public network. Our language is a variant of the pi calculus, with secure communications, mobile names, and high-level certificates, but without any explicit cryptography. Within this language, security properties can be conveniently studied using trace properties and observational equivalences, even in the presence of an arbitrary (abstract) adversary.
With some care, these security properties can be achieved in a concrete setting, relying on standard cryptographic primitives and computational assumptions, even in the presence of an adversary modeled as an arbitrary probabilistic polynomial-time algorithm. To this end, we develop a cryptographic implementation that preserves all properties for all safe programs. We give a series of soundness and completeness results that precisely relate the language to its implementation.

1 Secure Implementations of Communications Abstractions

When designing and verifying security protocols, some level of idealization is needed to provide manageable mathematical treatment. Accordingly, two views of cryptography have been developed 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 is also arguably 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.

In this work, we develop a first sound and complete implementation of a distributed process calculus. Our calculus is a variant of the pi calculus; it provides name mobility, reliable messaging and authentication primitives, but neither explicit cryptography nor probabilistic behaviors. Taking advantage of concurrency theory, it supports simple reasoning, based on labeled transitions and observational equivalence. We precisely define its concrete implementation in a computational setting. We establish general soundness and completeness results in the presence of active adversaries, for both trace properties and observational equivalences, essentially showing that high level reasoning accounts

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

To appear in the proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006), LNCS.

for all low-level adversaries. We illustrate our approach by coding security protocols and establishing their computational correctness by simple formal reasoning.

We implement high-level functionalities using cryptography, not high-level views of cryptographic primitives. Following recent related works, we could instead have proceeded in two steps, by first compiling high-level communications to an intermediate calculus with ideal, explicit cryptography (in the spirit of [3,2]), then establishing the computational soundness of this calculus with regards to computational cryptography. However, this second step is considerably more delicate than our present goal, inasmuch as one must provide a sound implementation for an arbitrary usage of ideal cryptography. In contrast, for instance, our language keeps all keys implicit, so no high-level program may ever leak a key or create an encryption cycle. (We considered targeting existing idealized cryptographic frameworks with soundness theorems, but their reuse turned out to be more complex than a direct implementation.)

Our 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 their properties and, at the same time, be as abstract as possible. In particular, our high-level environments should be given precisely the same capabilities as low-level probabilistic polynomial-time (PPT) adversaries. For example, our language supports abstract reliable messaging: message senders and receivers are authenticated, message content is protected, and messages are delivered at most once. On the other hand, under the conservative assumption that the adversary controls the network, we cannot guarantee message delivery, nor implement private channels (such that some communications may be undetected). Hence, the simple rule $\overline{c}\langle M\rangle.P \mid c(x).Q \rightarrow P \mid Q\{M/x\}$, which models silent communication "in the ether" for the pi calculus, is too abstract for our purposes. (For instance, if P and Q are implemented on different machines connected by a public network, and even if c is a restricted channel, the adversary can simply block all communications.) Instead, we design high-level rules for communications between explicit principals, mediated by an adversary, with abstract labels that enable the environment to perform traffic analysis but not forge messages or observe their payload. Similarly, process calculi feature non-deterministic infinite computations, and we need to curb these features to meet our low-level complexity requirements.

Contents This extended abstract is organized as follows. Section 2 defines our low-level target model. Section 3 presents our high-level language and semantics. Section 4 defines and illustrates high-level equivalences. Section 5 outlines our concrete implementation. Section 6 states our soundness and correctness theorems. Section 7 concludes.

A technical report [6] provides additional details and definitions, including the definition of our cryptographic implementation, examples and applications, and all proofs.

Related Work Within formal cryptography, process calculi are widely used to model security protocols. For example, the spi calculus of Abadi and Gordon [4] neatly models secret keys and fresh nonces using names and their dynamic scopes. Representing active attackers as pi calculus contexts, one can state (and prove) trace properties and observational equivalences that precisely capture the security goals for these protocols. Automated provers (e.g. [10]) also help verify these goals.

Abadi, Fournet, and Gonthier develop distributed implementations for variants of the join calculus, with high-level security but no cryptography, roughly comparable to our high-level language. Their implementation is coded within a lower-level calculus with formal cryptography. They establish full abstraction for observational equivalence [3,2]. Our approach is similar, but our implementation is considerably more concrete. Also, due to the larger distance between high-level processes and low-level machines, our soundness results are more demanding. Abadi and Fournet also propose a labeled semantics for traffic analysis, in the context of a pi calculus model of a fixed protocol for private authentication [1].

The computational soundness of formal cryptography is an active area of research, with many recent results for languages that include selected cryptographic primitives. Abadi and Rogaway initially consider formal encryption against passive attackers [5] and establish the soundness of indistinguishability. Backes, Pfitzmann and Waidner [8] achieve a first soundness result with active attackers, initially for public-key encryption and digital signatures. They extend their result to symmetric authentication [9] and encryption [7]. Micciancio and Warinschi [16] also establish soundness in the presence of active attacks, under different simpler assumptions.

Other works develop computationally sound implementations of more abstract security functions on top of cryptography. For example, Canetti and Krawczyk build computational abstractions of secure channels in the context of key exchange protocols, with modular implementations, and they establish sufficient conditions to realize these channels [11]. Targeting the idealized cryptographic model of Backes et al. [8], Laud [14] implements a deterministic process calculus and establishes the computational soundness of a type system for secrecy.

Another interesting approach is to supplement process calculi with concrete probabilistic or polynomial-time semantics. Unavoidably, reasoning on processes becomes more difficult. For example, Lincoln, Mitchell, Mitchell, and Scedrov [15] introduce a probabilistic process algebra for analyzing security protocols, such that parallel contexts coincide with probabilistic polynomial-time adversaries. In this framework, further extended by Mitchell, Ramanathan, Scedrov, and Teague [17], they develop an equational theory and bisimulation-based proof techniques.

2 Low-Level Target Model

Before presenting our language design and implementation, we specify the target systems. We rely on standard notions of security for cryptographic primitives (CCA2 for encryption [18], CMA for signing [13]) recalled in the technical report.

We consider systems that consist of a finite number of communicating principals $a, b, c, e, u, v, \ldots \in \mathsf{Prin}$. Each principal runs its own program, written in our high-level language and executed by the PPT machine outlined in Section 5. Each machine M_a has two wires, $\mathbf{in}_a$ and $\mathbf{out}_a$, representing a basic network interface. When activated, the machine reads a bitstring from $\mathbf{in}_a$, performs some local computation, then writes a bitstring on $\mathbf{out}_a$ and yields. The machine embeds probabilistic algorithms for encryption, signing, and random-number generation—thus the machine outputs are random variables. The machine is also parameterized by a security parameter $\eta \in \mathbb{N}$—intuitively, the length for all keys—thus these outputs are ensembles of probabilities.

Some of these machines may be corrupted, under the control of the attacker; their implementation is then unspecified and treated as part of the attacker. We let $a, b \in \mathcal{H}$ with $\mathcal{H} \subset \mathsf{Prin}$ range over principals that comply with our implementation, and let $\mathsf{M} = (\mathsf{M}_a)_{a \in \mathcal{H}}$ describe our whole system. Of course, when a interacts with $u \in \mathsf{Prin}$, its implementation M_a does not know whether $u \in \mathcal{H}$ or not.

The adversary, A, is a PPT algorithm that controls the network, the global scheduler, and some compromised principals. At each moment, only one machine is active: whenever an adversary delivers a message to a principal, this principal is activated, runs until completion, and yields an output to the adversary.

Definition 1 (Run). *A run of* A *and* M *with security parameter* $\eta \in \mathbb{N}$ *goes as follows:*

1. *key materials are generated for every principal* $a \in \mathsf{Prin}$*;*
2. *every* M_a *is activated with* 1^η*, the keys for* a*, and the public keys for all* $u \in \mathsf{Prin}$*;*
3. A *is activated with* 1^η*, the keys for* $e \in \mathsf{Prin} \setminus \mathcal{H}$*, and the public keys for* $a \in \mathcal{H}$*;*
4. A *performs a series of low-level exchanges:*
   - A *writes a bitstring on wire* $\mathbf{in}_a$ *and activates* M_a *for some* $a \in \mathcal{H}$*;*
   - *upon completion of* M_a*,* A *reads a bitstring on* $\mathbf{out}_a$*;*
5. A *returns a bitstring* s*, written* $s \longleftarrow \mathsf{A}[\mathsf{M}]$.

To study their security properties, we compare systems that consist of machines running on behalf of the same principals $\mathcal{H} \subseteq \mathsf{Prin}$, but with different internal programs and states. Intuitively, two systems are equivalent when no adversary, starting with the information normally given to the principals $e \in \mathsf{Prin} \setminus \mathcal{H}$, can distinguish between their two behaviors, except with negligible probability (written $\operatorname{neg}(\eta)$). This is the notion of *computational indistinguishability* introduced by Goldwasser and Micali [12]. Our goal is to develop a simpler, higher-level semantics that entails indistinguishability.

Definition 2. *Two systems* M^0 *and* M^1 *are indistinguishable, written* $\mathsf{M}^0 \approx \mathsf{M}^1$*, when for every PPT adversary* A*, we have* $|\Pr[1 \longleftarrow \mathsf{A}[\mathsf{M}^0]] - \Pr[1 \longleftarrow \mathsf{A}[\mathsf{M}^1]]| \leq \operatorname{neg}(\eta)$.

3 A Distributed Calculus with Principals and Authentication

We now present our high-level language. We successively define terms, patterns, processes, configurations, and systems. We then give their operational semantics. Although some aspects of the design are unusual, the resulting calculus is still reasonably abstract and convenient for distributed programming.

Syntax and Informal Semantics Let Name be a countable set of *names* disjoint from Prin. Let $\mathtt{f}$ range over a finite number of function symbols, each with a fixed arity $k \geq 0$. Terms and patterns are defined by the following grammar:

$V, W ::=$	Terms
$\quad x, y$	variable
$\quad m, n \in \mathsf{Name}$	name
$\quad a, b, e, u, v \in \mathsf{Prin}$	principal identity
$\quad \mathtt{f}(V_1, \ldots, V_k)$	constructed term (when $\mathtt{f}$ has arity k)
$T, U ::=$	Patterns
$\quad ?x$	variable (binds x)
$\quad T\ as\ ?x$	alias (binds x to the term that matches T)
$\quad V$	constant pattern
$\quad \mathtt{f}(T_1, \ldots, T_k)$	constructed pattern (when $\mathtt{f}$ has arity k)

Names and principals identities are atoms, or "pure names", which may be compared with one another but otherwise do not have any structure. Constructed terms represent structured data, much like algebraic data types in ML or discriminated unions in C. They can represent constants and tags (when $k = 0$), tuples, and formatted messages. As usual, we write $\mathtt{tag}$ and (V_1, V_2) instead of $\mathtt{tag}()$ and $\mathtt{pair}(V_1, V_2)$. Patterns are used for analyzing terms and binding selected subterms to variables. For instance, the pattern $(\mathtt{tag}, ?x)$ matches any pair whose first component is $\mathtt{tag}$ and binds x to its second component. We write _ for a variable pattern that binds a fresh variable.

Local processes represent the active state of principals, with the following grammar:

$P, Q, R ::=$	Local processes
$\quad V$	asynchronous output
$\quad (T).Q$	input (binds $bv(T)$ in Q)
$\quad *(T).Q$	replicated input (binds $bv(T)$ in Q)
$\quad \mathtt{match}\ V\ \mathtt{with}\ T\ \mathtt{in}\ Q\ \mathtt{else}\ Q'$	matching (binds $bv(T)$ in Q)
$\quad \nu n.P$	name restriction ("new", binds n in P)
$\quad P \mid P'$	parallel composition
$\quad \mathbf{0}$	inert process

The asynchronous output V is just a pending message; its data structure is explained below. The input $(T).Q$ waits for an output that matches T then runs Q with the bound variables of T substituted by the matching subterms of the output message. The replicated input $*(T).Q$ behaves similarly but it can consume any number of outputs that match T and fork a copy of Q for each of them. The match process runs Q if V matches T, and runs Q' otherwise. The name restriction creates a fresh name n then runs P. Parallel composition represents processes that run in parallel, with the inert process $\mathbf{0}$ as unit. Free and bound names and variables for terms, patterns, and processes are defined as usual: x is bound in T if $?x$ occurs in T; n is bound in $\nu n.P$; x is free in T if it occurs in T and is not bound in T. An expression is closed when it has no free variables; it may have free names.

Our language features two forms of authentication, represented as constructors plus well-formed conditions on their usage in processes. Due to space constraints, this extended abstract only describes message authentication—the technical report also describes high level certificates that provide transferable data authentication.

Authenticated messages between principals are represented as terms of the form $\mathtt{auth}(V_1, V_2, V_3)$, written $V_1{:}V_2\langle V_3\rangle$, where V_1 is the sender, V_2 the receiver, and V_3 the content. We let M and N range over messages. The message M is *from* a (respectively *to* a) if a is the sender (respectively the receiver) of M. Authenticated messages are delivered at most once, to their designated receiver. As an example, $a{:}b\langle\texttt{Hello}\rangle$ is an (authentic) message from a to b with content `Hello`, a constructor with arity 0.

Finally, configurations represent assemblies of communicating principals, with the following grammar:

$C ::=$	configurations
$a[P]$	principal a with local state P
M/i	intercepted message M with index i
$C \mid C'$	distributed parallel composition
$\nu n.C$	name restriction ("new", binds n in C)

A configuration is an assembly of running principals, each with its own local state, plus an abstract record of the messages intercepted by the environment and not forwarded yet to their intended recipients. A system S is a top-level configuration (plus an abstract record of the certificates available to the adversary, omitted here).

We rely on well-formed conditions. In local processes, P is *well-formed for* $a \in \mathsf{Prin}$ when no pattern used for input in P matches any message from a. This condition prevents that messages sent by P be read back by some local input. In configurations, intercepted messages have distinct indices i and closed content M; principals have distinct identities a and well-formed local processes P_a. In systems, let $\mathcal{H}$ be the set of identities for all defined principals, called *compliant principals*; intercepted messages are from a to b for some $a, b \in \mathcal{H}$ with $a \neq b$.

Operational Semantics—Local Reductions We define our high-level semantics in two stages: local reductions between processes, then global labeled transitions between systems and their (adverse) environment. Processes, configurations, and systems are considered up to renaming of bound names and variables.

Structural equivalence, written $P \equiv P'$, represents structural rearrangements for local processes. As in the pi calculus, it is defined as the smallest congruence such that $P \equiv P \mid 0$, $P \mid Q \equiv Q \mid P$, $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$, $(\nu n.P) \mid Q \equiv \nu n.(P \mid Q)$ when $n \notin fn(Q)$, $\nu m.\nu n.P \equiv \nu n.\nu m.P$, and $\nu n.\mathbf{0} \equiv \mathbf{0}$. Intuitively, structural rearrangements are not observable (although this is quite hard to implement).

Local reduction step, written $P \rightarrow P'$, represents internal computation between local processes. It is defined as the smallest relation such that

(LComm)	$(T).Q \mid T\sigma \rightarrow Q\sigma$
(LRepl)	$*(T).Q \mid T\sigma \rightarrow Q\sigma \mid *(T).Q$
(LMatch)	$\texttt{match}\ T\sigma\ \texttt{with}\ T\ \texttt{in}\ P\ \texttt{else}\ Q \rightarrow P\sigma$
(LNoMatch)	$\texttt{match}\ V\ \texttt{with}\ T\ \texttt{in}\ P\ \texttt{else}\ Q \rightarrow Q$ when $V \neq T\sigma$ for any σ

(LParCtx) $\dfrac{P \rightarrow Q}{P \mid R \rightarrow Q \mid R}$ (LNewCtx) $\dfrac{P \rightarrow Q}{\nu n.P \rightarrow \nu n.Q}$ (LStruct) $\dfrac{P \equiv P' \quad P' \rightarrow Q' \quad Q' \equiv Q}{P \rightarrow Q}$

where σ ranges over substitutions of closed terms for the variables bound in T. The local process P is *stable* when it has no local reduction step, written $P \not\rightarrow$. We write $P \twoheadrightarrow Q$ when $P \rightarrow^{*}\equiv Q$ and $Q \not\rightarrow$.

Operational Semantics—System Transitions We define a labeled transition semantics for systems. Each labeled transition, written $S \xrightarrow{\gamma} S'$, represents a single interaction with the adversary. We let α and β range over input and output labels (respectively from and to the adversary), let γ range over labels, and let φ range over series of labels. We write $S \xrightarrow{\varphi} S'$ for a series of transitions with labels φ. Labeled transitions are defined by the following rules on configurations:

$$\text{(CFGOUT)}\frac{u \neq a}{a[a{:}u\langle V\rangle \mid Q] \xrightarrow{a:u\langle V\rangle} a[Q]} \qquad \text{(CFGIN)}\frac{u{:}a\langle V\rangle \mid P \twoheadrightarrow Q \quad u \neq a}{a[P] \xrightarrow{(u:a\langle V\rangle)} a[Q]}$$

$$\text{(CFGBLOCK)}\frac{C \xrightarrow{b:a\langle V\rangle} C' \quad i \text{ not in } C}{C \mid a[P] \xrightarrow{\nu i.b:a} C' \mid b{:}a\langle V\rangle/i \mid a[P]} \qquad \text{(CFGFWD)}\frac{C \xrightarrow{(M)} C'}{C \mid M/i \xrightarrow{(i)} C'}$$

$$\text{(CFGPRINCTX)}\frac{C \xrightarrow{\gamma} C' \quad \gamma \text{ not from/to } a}{C \mid a[P] \xrightarrow{\gamma} C' \mid a[P]} \qquad \text{(CFGMSGCTX)}\frac{C \xrightarrow{\gamma} C' \quad i \text{ not in } \gamma}{C \mid M/i \xrightarrow{\gamma} C' \mid M/i}$$

$$\text{(CFGOPEN)}\frac{C \xrightarrow{\beta} C' \quad n \text{ free in } \beta}{\nu n.C \xrightarrow{\nu n.\beta} C'} \qquad \text{(CFGNEWCTX)}\frac{C \xrightarrow{\gamma} C' \quad n \text{ not in } \gamma}{\nu n.C \xrightarrow{\gamma} \nu n.C'}$$

$$\text{(CFGSTR)}\frac{C \equiv D \quad D \xrightarrow{\gamma} D' \quad D' \equiv C'}{C \xrightarrow{\gamma} C'}$$

where structural equivalence on configurations, written $C \equiv C'$, is defined by the same rules as for processes plus Rule $\nu n.a[P] \equiv a[\nu n.P]$.

Rules (CFGOUT) and (CFGIN) represent "intended" interactions with the environment, as usual. They enable local processes to send messages to other principals, and to receive their messages. The transition label conveys the complete message content.

Rules (CFGBLOCK) and (CFGFWD) reflect the actions of an active attacker that intercepts, then selectively forwards, messages exchanged between compliant principals; unlike the (COMM) rule of the pi calculus, they ensure that the environment mediates all communications between principals. The label produced by (CFGBLOCK) signals the message interception; the label conveys partial information on the message content that can be observed from its wire format: the environment learns that an opaque message is sent by b, with intended recipient a. In addition, the intercepted message content is recorded within the configuration, using a fresh index i. Later on, when the environment performs an input with label (i), Rule (CFGFWD) restores the original message content and consumes M/i; this ensures that intercepted messages are delivered at most once.

The local-reduction hypothesis in Rules (CFGIN) makes local computations atomic, as they must complete immediately upon receiving a message and lead to some updated stable process Q. Intuitively, this enforces a transactional semantics for local steps, and prevents any observation of their transient internal state. (Otherwise, the environment may for instance observe the order of appearance of outgoing messages.) On the other hand, any outgoing messages are kept within Q; the environment can obtain all of them via rules (CFGOUT) and (CFGBLOCK) at any time, since those outputs commute with any subsequent transitions.

The rest of the rules for configurations are standard closure rules with regards to contexts and structural rearrangements: Rule (CFGOPEN) is the scope extrusion rule of the pi calculus that opens the scope of a restricted name included in a message sent to the environment. In contrast with intercepted messages, messages sent to a principal not defined in the configuration are transmitted unchanged to the environment, after applying the context rules. In Rule (CFGPRINCTX), condition γ not from a excludes inputs from the environment that forge a message from a, whereas condition γ not to a excludes outputs that may be transformed by Rule (CFGBLOCK).

We define auxiliary notions of transitions, used to describe our implementation. We say that S is *stable* when all local processes are stable and S has no output transition. (Informally, S is waiting for any input from the environment.) We say that a series of transitions $S \xrightarrow{\varphi} S'$ is *normal* when every input is followed by a maximal series of outputs leading to a stable system, that is, $\varphi = \varphi_1\varphi_2 \ldots \varphi_n$, $\varphi_i = \alpha_i\widetilde{\beta}_i$, and $S = S_0 \xrightarrow{\varphi_1} S_1 \xrightarrow{\varphi_2} S_2 \ldots \xrightarrow{\varphi_n} S_n = S'$ for some stable systems $S_0, \ldots, S_n$.

By design, our semantics is compositional, as its rules are inductively defined on the structure of configurations. For instance, we obtain that interactions with a principal that is implicitly controlled by the environment are *at least* as expressive as those with any principal explicited within the system.

4 High-Level Equivalences and Safety

Now that we have labeled transitions that capture our implementation constraints, we can apply standard definitions and proof techniques from concurrency theory to reason about systems. Our computational soundness results are useful (and non-trivial) inasmuch as transitions are simpler and more abstract than low-level adversaries. In addition to trace properties (used, for instance, to express authentication properties as correspondences between transitions) , we consider equivalences between systems.

Intuitively, two systems are equivalent when their environment observes the same transitions. Looking at immediate observations, we say that two systems S_1 and S_2 *have the same labels* when, if $S_1 \xrightarrow{\gamma} S_1'$ for some S_1' (and the name exported by γ are not free in S_2), then $S_2 \xrightarrow{\gamma} S_2'$ for some S_2', and vice versa. More generally, bisimilarity demands that this remains the case after matching transitions:

Definition 3 (Bisimilarity). *The relation $\mathcal{R}$ on systems is a labeled simulation when, for all $S_1 \mathrel{\mathcal{R}} S_2$, if $S_1 \xrightarrow{\gamma} S_1'$ (and the names exported by γ are not free in S_2) then $S_2 \xrightarrow{\gamma} S_2'$ and $S_1' \mathrel{\mathcal{R}} S_2'$. Labeled bisimilarity, written $\approx$, is the largest symmetric labeled simulation.*

In particular, if $S \approx S'$, then S and S' define the same principals and have the same intercepted-message indices. We also easily verify some congruence properties: our equivalence is preserved by name restrictions, definitions of additional principals, and deletions of intercepted messages.

Lemma 1. *1. If $C_1 \approx C_2$, then $\nu n.C_1 \approx \nu n.C_2$.*
2. If $C_1 \approx C_2$, then $C_1 \mid a[P] \approx C_2 \mid a[P]$ if these systems are well-formed.
3. If $\nu\widetilde{n}_1.(C_1 \mid M_1/i) \approx \nu\widetilde{n}_2.(C_2 \mid M_2/i)$, then $\nu\widetilde{n}_1.C_1 \approx \nu\widetilde{n}_2.C_2$.

As we quantify over all local processes, we must at least bound their computational power. Indeed, our language is expressive enough to code Turing machines and, for instance, one can easily write a local process that receives a high-level encoding of the security parameter η (e.g. as a series of η messages) then delays a message output by 2^η reduction steps, or even implements an 'oracle' that performs some brute-force attacks using high level implementations of cryptographic algorithms.

Similarly, we must restrict non-deterministic behaviors. Process calculi often feature non-determinism as a convenience when writing specifications, to express uncertainty as regards the environment. Sources of non determinism include local scheduling, hidden in the associative-commutative laws for parallel composition, and internal choices. Accordingly, abstract properties and equivalences typically only consider the existence of transitions—not their probability. Observable non-determinism is problematic in a computational cryptographic setting, as for instance a non-deterministic process may be used as an oracle to guess every bit of a key in linear time.

We arrive at the following definitions. We let $\lceil\cdot\rceil$ compute the (high level) size of systems, labels, and transitions, with for instance $\lceil S \xrightarrow{\gamma} S'\rceil = \lceil S\rceil + \lceil\gamma\rceil + \lceil S'\rceil + 1$, and let $input(\varphi)$ be the input labels of φ.

Definition 4 (Safe Systems). *A system S is* polynomial *when there exists a polynomial p such that, for any φ, if $S \xrightarrow{\varphi} S'$ then $\lceil S \xrightarrow{\varphi} S'\rceil \leq p(\lceil input(\varphi)\rceil)$.*

A system S is safe *when it is polynomial and, for any φ, if $S \xrightarrow{\varphi} S_1$ and $S \xrightarrow{\varphi} S_2$ then S_1 and S_2 have the same labels.*

Hence, starting from a safe process, a series of labels fully determines any further observation. Safety is preserved by all transitions, and also uniformly bounds (for example) the number of local reductions and new names.

These restrictions are serious, but they are also easily established when writing simple programs and protocols. (Still, it would be interesting to relax them, maybe using a probabilistic process calculus.) Accordingly, our language design prevents trivial sources of non-determinism and divergence (e.g. with pattern matching on values, and replicated inputs instead of full-fledged replication); further, most internal choices can be coded as external choices driven by the inputs of our abstract environment.

We can adapt usual bisimulation proof techniques to establish both equivalences and safety: instead of examining all series of labels φ, it suffices to examine single transitions for the systems in the candidate relation.

Lemma 2 (Bisimulation Proof). *Let $\mathcal{R}$ be a reflexive labeled bisimulation such that, for all related systems $S_1 \mathcal{R} S_2$, if $S_1 \xrightarrow{\gamma} S_1'$ and $S_2 \xrightarrow{\gamma} S_2'$, then $S_1' \mathcal{R} S_2'$.*

Polynomial systems related by $\mathcal{R}$ are safe and bisimilar.

We illustrate our definitions using basic examples of secrecy and authentication stated as equivalences between a protocol and its specification (adapted from [2]). Consider a principal a that sends a single message. In isolation, we have the equivalence $a[a{:}b\langle V\rangle] \approx a[a{:}b\langle V'\rangle]$ if and only if $V = V'$, since the environment observes V on the label of the transition $a[a{:}b\langle V\rangle] \xrightarrow{a:b\langle V\rangle} a[\mathbf{0}]$.

Consider now the system $S(V,W) = a[a{:}b\langle V,W\rangle] \mid b[(a{:}\langle ?x, \_\rangle).P]$, with an explicit process for principal b that receives a's message and, assuming the message is a pair, runs P with the first element of the pair substituted for x. For any terms W_1 and W_2, we have $S(V,W_1) \approx S(V,W_2)$. This equivalence states the strong secrecy of W, since its value cannot affect the environment. The system has two transitions $S(V,W) \xrightarrow{\nu i.a:b} \xrightarrow{(i)} a[\mathbf{0}] \mid b[P\{V/x\}]$.

Further, the equivalence $S(V,W) \approx a[a{:}b\langle\rangle] \mid b[(a{:}\langle\_\rangle).P\{V/x\}]$ captures both the authentication of V and the absence of observable information on V and W in the communicated message, since the protocol $S(V,W)$ behaves just like another protocol that sends a dummy message instead of V, W.

5 A Concrete Implementation (Outline)

We systematically map high-level systems S to the machines of Section 2, mapping each principal $a[P_a]$ of S to a PPT machine M_a that executes P_a. Due to space constraints, we only give an outline of our implementation, defined in the technical report. The implementation mechanisms are simple, but they need to be carefully specified and composed. (As a non-trivial example, when a machine outputs several messages, possibly to the same principals, we must sort the messages after encryption so that their ordering on the wire leaks no information on the computation that produced them.)

We use two concrete representations for terms: a wire format for (signed, encrypted) messages between principals, and an internal representation for local terms. Various bitstrings represent constructors, principal identities, names, and certificates. Marshaling and unmarshaling functions convert between internal and wire representations. When marshaling a locally restricted name n for the first time, we draw a bitstring s of length η uniformly at random, associate it with n, and use it to represent n on the wire. When unmarshaling a bitstring s into a name, if s is not associated with any local name, we create a new internal identifier n for the name, and also associate s with n.

Local processes are represented in normal form for structural equivalence, using internal terms and multisets of local inputs, local outputs, and outgoing messages. We implement reductions using an abstract machine that matches inputs and outputs using an arbitrary deterministic, polynomial-time scheduler.

To keep track of the runtime state for our machines, we supplement high-level systems S with *shadow states* D that record sufficient information so that each machine is a function $\mathsf{M}_a(S, \mathsf{D})$. For instance, D records maps from names and intercepted messages to bitstrings, and from principals to their keys and the content of their anti-replay caches. The shadow D also determines the information available to the attacker, coded as a bitstring $public(\mathsf{D})$. The structure of $public(\mathsf{D})$ sets the interface between attackers and low-level systems, called the *shape* of D. For instance, the shape fixes the free names that may occur in S, and $public(\mathsf{D})$ provides their associated bitstrings.

In general, a system S may contain restricted names shared between local processes and intercepted messages, making it non-trivial to describe a concrete initialization mechanism that produces $\mathsf{M}(S, \mathsf{D})$ and $public(\mathsf{D})$. Instead of explicitly coding low-level initialization, we define it as the run of a high-level initialization protocol $S^\circ \xrightarrow{\varphi^\circ} S$ that lets the principals exchange names and yield intercepted messages to the environment. In the initialization protocol, S° is a system with no intercepted messages and no free names in local processes. For any system S, there are such transitions $S^\circ \xrightarrow{\varphi^\circ} S$ and, applying a variant of Theorem 1, there is a PPT algorithm $\mathsf{A}_{\varphi^\circ}$ that simulates φ° and produces $public(\mathsf{D})$ from some $public(\mathsf{D}^\circ)$, where D° is the shadow produced by Definition 1(1–3). Thus, we define a run of $\mathsf{M}(S, \mathsf{D})$ with adversary A, written $\mathsf{A}[\mathsf{M}(S,\mathsf{D})]$, as a run of $(\mathsf{A}_{\varphi^\circ};\mathsf{A})[\mathsf{M}(S^\circ,\mathsf{D}^\circ)]$ where $\mathsf{A}_{\varphi^\circ};\mathsf{A}$ first runs $\mathsf{A}_{\varphi^\circ}$ then starts A with input $public(\mathsf{D})$. We then say that D is a valid shadow for S.

6 Soundness and Completeness Results

In this section we show that properties that hold with the high-level semantics can be carried over to the low-level implementation, and the other way around. Due to space constraints, most auxiliary results and all proofs appear in the technical report [6].

Our first theorem expresses the soundness of the high-level operational semantics: every series of transitions can be executed (and checked) by a low-level attacker. Said otherwise, the high-level semantics does not give too much power to the environment.

Theorem 1. *For any shape of* D *and labels* φ*, there is a PPT algorithm* A_φ *such that, for any safe stable system* S *with valid shadow* D *where the new names of* φ *are not free in* D*, one of the following holds with overwhelming probability:*

- $1 \longleftarrow \mathsf{A}_\varphi[\mathsf{M}(S,\mathsf{D})]$ *and there exists* S' *with normal transitions* $S \xrightarrow{\varphi} S'$*; or*
- $0 \longleftarrow \mathsf{A}_\varphi[\mathsf{M}(S,\mathsf{D})]$ *and there are no normal transitions* $S \xrightarrow{\varphi} S'$*.*

Since we can characterize any trace using an adversary, we also obtain completeness for trace equivalence: low-level equivalence implies high-level trace equivalence.

Theorem 2. *Let* S_1 *and* S_2 *be safe stable systems with valid shadow* D *such that* $\mathsf{M}(S_1,\mathsf{D}) \approx \mathsf{M}(S_2,\mathsf{D})$*. If there are normal transitions* $S_1 \xrightarrow{\varphi} S_1'$ *and the new names of* φ *are not free in* D*, then there are normal transitions* $S_2 \xrightarrow{\varphi} S_2'$*.*

Our next theorem expresses the completeness of our high-level transitions: every low-level attack can be described in terms of high-level transitions. More precisely, the probability that an interaction with a PPT adversary yields a machine state unexplained by any high-level transitions is negligible.

Theorem 3. *Let* S *be a safe stable system with valid shadow* D *and* A *a PPT algorithm.*
The probability that $\mathsf{A}[\mathsf{M}(S,\mathsf{D})]$ *completes and leaves the system in state* M' *with* $\mathsf{M}' \neq \mathsf{M}(S',\mathsf{D}')$ *for any normal transitions* $S \xrightarrow{\varphi} S'$ *with valid shadow* D' *is negligible.*

Finally, our main result states the soundness of equivalence: to show that two stable systems are indistinguishable, it suffices to show that they are safe and bisimilar.

Theorem 4. *Let* S_1 *and* S_2 *be safe stable systems with valid shadow* D*.*
If $S_1 \approx S_2$*, then* $\mathsf{M}(S_1,\mathsf{D}) \approx \mathsf{M}(S_2,\mathsf{D})$*.*

7 Conclusions and Future Work

We designed a simple, abstract language for secure distributed communications. Our language provides uniform protection for all messages; it is expressive enough to program a large class of protocols; it also enables simple reasoning about security properties in the presence of active attackers, using labeled traces and equivalences. We implemented this calculus as a collection of concrete PPT machines embedding standard cryptographic algorithms, and established that low-level PPT adversaries that control their scheduling and the network have essentially the same power as (much simpler) high-level environments. To the best of our knowledge, these are the first cryptographic soundness and completeness results for a distributed process calculus.

We also identified and discussed difficulties that stem from the discrepancy between the two models, and developed proofs that combine techniques from process calculi and cryptography. It would be interesting (and hard) to extend the expressiveness of our calculus, for instance with secrecy and probabilistic choices.

Acknowledgments This paper benefited from discussions with Martín Abadi, Tuomas Aura, Karthik Bhargavan, Andy Gordon, and David Pointcheval.

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.
2. M. Abadi, C. Fournet, and G. Gonthier. Authentication primitives and their compilation. In *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 A. D. Gordon. A calculus for cryptographic protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, 1999.
5. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
6. P. Adão and C. Fournet. Cryptographically sound implementations for communicating processes. Technical report MSR-TR-2006-49. Microsoft Research, 2006.
7. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *CSFW-17*, pages 204–218. IEEE, 2004.
8. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *CCS 2003*, pages 220–230. ACM, 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, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. In *LICS 2005*, pages 331–340. IEEE, 2005.
11. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT 2001*, LNCS 2045, pages 453–474. Springer, 2001.
12. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and Systems Sciences*, 28(2):270–299, 1984.
13. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attack. *SIAM Journal on Computing*, 17(2):281–308, 1988.
14. P. Laud. Secrecy types for a simulatable cryptographic library. In *CCS 2005*, pages 26–35. ACM, 2005.
15. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *CCS 1998*, pages 112–121. ACM, 1998.
16. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *TCC 2004*, LNCS 2951, pages 133–151. Springer, 2004.
17. J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for the analysis of cryptographic protocols. *Theoretical Computer Science*, 2006.
18. C. Rackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO'91*, LNCS 576, pages 433–444. Springer, 1991.