A Computationally Sound Mechanized Prover for Security Protocols

Bruno Blanchet

CNRS, École Normale Supérieure, Paris
blanchet@di.ens.fr

Abstract. We present a new mechanized prover for secrecy properties of cryptographic protocols. In contrast to most previous provers, our tool does not rely on the Dolev-Yao model, but on the computational model. It produces proofs presented as sequences of games; these games are formalized in a probabilistic polynomial-time process calculus. Our tool provides a generic method for specifying security properties of the cryptographic primitives, which can handle shared- and public-key encryption, signatures, message authentication codes, and hash functions. Our tool produces proofs valid for a number of sessions polynomial in the security parameter, in the presence of an active adversary. We have implemented our tool and tested it on a number of examples of protocols from the literature.

1 Introduction

There exist two main frameworks for studying cryptographic protocols. In the computational model, messages are bitstrings, and the adversary is a probabilistic polynomial-time Turing machine. This model is close to the real execution of protocols, but the proofs are usually manual and informal. In contrast, in the formal, Dolev-Yao model, cryptographic primitives are considered as perfect blackboxes, modeled by function symbols in an algebra of terms, possibly with equations. The adversary can compute using these blackboxes. This abstract model makes it possible to build automatic verification tools, but the security proofs are in general not sound with respect to the computational model.

Since the seminal paper by Abadi and Rogaway [3], there has been much interest in relating both frameworks (see for example [1, 8, 11, 20, 24, 25, 34, 35]), to show the soundness of the Dolev-Yao model with respect to the computational model, and thus obtain automatic proofs of protocols in the computational model. However, this approach has limitations: since the computational and Dolev-Yao models do not correspond exactly, additional hypotheses are necessary in order to guarantee soundness. (For example, key cycles have to be excluded, or a specific security definition of encryption is needed [5].)

In this paper, we propose a different approach for automatically proving protocols in the computational model: we have built a mechanized prover that works directly in the computational model, without considering the Dolev-Yao model. Our tool produces proofs valid for a number of sessions polynomial in the security parameter, in the presence of an active adversary. These proofs are presented as sequences of games, as used by cryptographers [15, 41, 42]: the initial game represents the protocol to prove; the goal is to show that the probability of breaking a certain security property (secrecy in this paper) is negligible in this game; intermediate games are obtained each from the previous one by transformations such that the difference of probability between consecutive games is negligible; the final game is such that the desired probability is obviously negligible from the form of the game. The desired probability is then negligible in the initial game.

We represent games in a process calculus. This calculus is inspired by the pi-calculus, and by the calculi of [30, 31, 36] and of [29]. In this calculus, messages are bitstrings, and cryptographic primitives are functions from bitstrings to bitstrings. The calculus has a probabilistic semantics, and all processes run in polynomial time. The main tool for specifying security properties is observational equivalence: Q is observationally equivalent to Q', $Q \approx Q'$, when the adversary has a negligible probability of distinguishing Q from Q'. With respect to previous calculi mentioned above, our calculus introduces an important novelty which is key for the automatic proof of cryptographic protocols: the values of all variables during the execution of a process are stored in arrays. For instance, $x[i]$ is the value of x in the i-th copy of the process that defines x. Arrays replace lists often used by cryptographers in their manual proofs of protocols. For example, consider the definition of security of a message authentication code (*mac*). Informally, this definition says that the adversary has a negligible probability of forging a *mac*, that is, that all correct macs have been computed by calling the mac oracle. So, in cryptographic proofs, one defines a list containing the arguments of calls to the mac oracle, and when checking a mac of a message m, one can additionally check that m is in this list, with a negligible change in probability. In our calculus, the arguments of the mac oracle are stored in arrays, and we perform a lookup in these arrays in order to find the message m. Arrays make it easier to automate proofs since they are always present in the calculus: one does not need to add explicit instructions to insert values in them, in contrast to the lists used in manual proofs. Therefore, many trivially sound but difficult to automate syntactic transformations disappear.

Our prover relies on a collection of game transformations, in order to transform the initial protocol into a game on which the desired security property is obvious. The most important kind of transformations comes from the definition of security of cryptographic primitives. As described in Section 3.3, these transformations can be specified in a generic way: we represent the definition of security of each cryptographic primitive by an observational equivalence $L \approx R$, where the processes L and R encode functions: they input the arguments of the function and send its result back. Then, the prover can automatically transform a process Q that calls the functions of L (more precisely, contains as subterms terms that perform the same computations as functions of L) into a process Q' that calls the functions of R instead. We have used this technique to specify several variants of shared- and public-key encryption, signature, message authentication codes, and hash functions, simply by giving the appropriate equivalence $L \approx R$ to the prover. Other game transformations are syntactic transformations, used in order to be able to apply the definition of cryptographic primitives, or to simplify the game obtained after applying these definitions.

In order to prove protocols, these game transformations are organized using a proof strategy based on advice: when a transformation fails, it suggests other transformations that should be applied before, in order to enable the desired transformation. Thanks to this strategy, protocols can often be proved in a fully automatic way. For delicate cases, our prover has an interactive mode, in which the user can manually specify the transformations to apply. It is usually sufficient to specify a few transformations coming from the security definitions of primitives, by indicating the concerned cryptographic primitive and the concerned secret key if any; the prover infers the intermediate syntactic transformations by the advice strategy. This mode is helpful for proving some public-key protocols, in which several security definitions of primitives can be applied, but only one leads to a proof of the protocol. Importantly, our prover is always sound: whatever indications the user gives, when the prover shows a security property of the protocol, the property indeed holds assuming the given hypotheses on the cryptographic primitives.

Our prover has been implemented in Ocaml (9700 lines of code) and is available at `http://www.di.ens.fr/~blanchet/cryptoc-eng.html`.

Related Work Results that show the soundness of the Dolev-Yao model with respect to the computational model, e.g. [20, 25, 35], make it possible to use Dolev-Yao provers in order to prove protocols in the computational model. However, these results have limitations, in particular in terms of allowed cryptographic primitives (they must satisfy strong security properties so that they correspond to Dolev-Yao style primitives), and they require some restrictions on protocols (such as the absence of key cycles).

Several frameworks exist for formalizing proofs of protocols in the computational model. Backes, Pfitzmann, and Waidner [6, 8, 9] have designed an abstract cryptographic library including symmetric and public-key encryption, message authentication codes, signatures, and nonces and shown its soundness with respect to computational primitives, under arbitrary active attacks. Backes and Pfitzmann [7] relate the computational and formal notions of secrecy in the framework of this library. Recently, this framework has been used for a computationally-sound machine-checked proof of the Needham-Schroeder-Lowe protocol [43]. Canetti [18] introduced the notion of universal composability. With Herzog [19], they show how a Dolev-Yao-style symbolic analysis can be used to prove security properties of protocols within the framework of universal composability, for a restricted class of protocols using public-key encryption as only cryptographic primitive. Then, they use the automatic Dolev-Yao verification tool Proverif [16] for verifying protocols in this framework. Lincoln, Mateus, Mitchell, Mitchell, Ramanathan, Scedrov, and Teague [30, 31, 33, 36, 40] developed a probabilistic polynomial-time calculus for the analysis of cryptographic protocols. They define a notion of process equivalence for this calculus, derive compositionality properties, and define an equational proof system for this calculus. Datta, Derek, Mitchell, Shmatikov, and Turuani [21] have designed a computationally sound logic that enables them to prove computational security properties using a logical deduction system. These frameworks can be used to prove security properties of protocols in the computational sense, but except for [19] which relies on a Dolev-Yao prover, they have not been mechanized up to now, as far as we know.

Laud [27] designed an automatic analysis for proving secrecy for protocols using shared-key encryption, with passive adversaries. He extended it [28] to active adversaries, but with only one session of the protocol. This work is the closest to ours. We extend it considerably by handling more primitives, and a polynomial number of sessions.

Recently, Laud [29] designed a type system for proving security protocols in the computational model. This type system handles shared- and public-key encryption, with an unbounded number of sessions. This system relies on the Backes-Pfitzmann-Waidner library. Type inference has not been implemented yet, and we believe that it would not be obvious to automate.

Barthe, Cerderquist, and Tarento [10, 44] have formalized the generic model and the random oracle model in the interactive theorem prover Coq, and proved signature schemes in this framework. In contrast to our specialized prover, proofs in generic interactive theorem provers require a lot of human effort, in order to build a detailed enough proof for the theorem prover to check it.

Halevi [23] explains that implementing an automatic prover based on sequences of games would be useful, and suggests ideas in this direction, but does not actually implement one.

Outline The next section presents our process calculus for representing games. Section 3 describes the game transformations that we use for proving protocols. Section 4 gives criteria for proving secrecy properties of protocols. Section 5 explains how the prover chooses which transformation to apply at each point. Section 6 presents our experimental results, and Section 7 concludes. The appendix contains details on the modeling of some cryptographic primitives and proof sketches of our results.

Notations We recall the following standard notations. We denote by $\{M_1/x_1, \ldots, M_m/x_m\}$ the substitution that replaces x_j with M_j for each $j \leq m$. The cardinal of a set or multiset S is denoted $|S|$. We use $\uplus$ for multiset union. When S is a multiset, $S(x)$ is the number of elements of S equal to x. When S and S' are multisets, $\max(S, S')$ is the multiset such that $\max(S, S')(x) = \max(S(x), S'(x))$. If S is a finite set, $x \overset{R}{\leftarrow} S$ chooses a random element uniformly in S and assigns it to x. If $\mathcal{A}$ is a probabilistic algorithm, $x \leftarrow \mathcal{A}(x_1, \ldots, x_m)$ denotes the experiment of choosing random coins r and assigning to x the result of running $\mathcal{A}(x_1, \ldots, x_m)$ with coins r. Otherwise, $x \leftarrow M$ is a simple assignment statement.

2 A Calculus for Games

2.1 Syntax and Informal Semantics

The syntax of our calculus is summarized in Figure 1. We denote by η the security parameter, which determines in particular the length of keys.

$M, N ::=$	terms
i	replication index
$x[M_1, \ldots, M_m]$	variable access
$f(M_1, \ldots, M_m)$	function application
$Q ::=$	input process
0	nil
$Q \mid Q'$	parallel composition
$!^{i \leq n} Q$	replication n times
$\mathit{newChannel}\ c; Q$	restriction for channels
$c[M_1, \ldots, M_l](x_1[i_1, \ldots, i_m] : T_1, \ldots, x_k[i_1, \ldots, i_m] : T_k); P$	input
$P ::=$	output process
$\overline{c[M_1, \ldots, M_l]}\langle N_1, \ldots, N_k \rangle; Q$	output
$\mathit{new}\ x[i_1, \ldots, i_m] : T; P$	random number generation (uniform)
$\mathit{let}\ x[i_1, \ldots, i_m] : T = M\ \mathit{in}\ P$	assignment
$\mathit{if}\ M\ \mathit{then}\ P\ \mathit{else}\ P'$	conditional
$\mathit{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathit{suchthat}\ \mathit{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathit{then}\ P_j)\ \mathit{else}\ P$	array lookup

Fig. 1. Syntax of the process calculus

This calculus assumes a countable set of channel names, denoted by c. There is a mapping maxlen_η from channels to integers, such that $\mathrm{maxlen}_\eta(c)$ is the maximum length of a message sent on channel c. Longer messages are truncated. For all c, $\mathrm{maxlen}_\eta(c)$ is polynomial in η. (This is key to guaranteeing that all processes run in probabilistic polynomial time.)

Our calculus also assumes a set of parameters, denoted by n, which correspond to integer values polynomial in the security parameter, so we define $I_\eta(n) = q(\eta)$ where q is a polynomial, and $I_\eta(n)$ denotes the interpretation of n for a given value of the security parameter η.

Our calculus also assumes a set of types, denoted by T. For each value of the security parameter η, each type corresponds to a subset $I_\eta(T)$ of *Bitstring* $\cup \{\perp\}$ where *Bitstring* is the set of all bitstrings and $\perp$ is a special symbol. The set $I_\eta(T)$ must be recognizable in polynomial time, that is, there exists an algorithm that decides whether $x \in I_\eta(T)$ in time polynomial in the length of x and the value of η. Let *fixed-length* types be types T such that $I_\eta(T)$ is the set of all bitstrings of a certain length, this length being a function of η bounded by a polynomial. Let *large* types be types T such that $\frac{1}{|I_\eta(T)|}$ is negligible. ($f(\eta)$ is *negligible* when for all polynomials q, there exists $\eta_o \in \mathbb{N}$ such that for all $\eta > \eta_0$, $f(\eta) \leq \frac{1}{q(\eta)}$.) Particular types are predefined: *bool*, such that $I_\eta(\mathit{bool}) = \{0, 1\}$, where 0 means false and 1 means true; *bitstring*, such that $I_\eta(\mathit{bitstring}) = \mathit{Bitstring}$; $\mathit{bitstring}_\perp$ such that $I_\eta(\mathit{bitstring}_\perp) = \mathit{Bitstring} \cup \{\perp\}$; $[1, n]$ where n is a parameter, such that $I_\eta([1, n]) = [1, I_\eta(n)]$. (We consider integers as bitstrings without leading zeroes.)

The calculus also assumes a finite set of function symbols f. Each function symbol comes with a type declaration $f : T_1 \times \ldots \times T_m \rightarrow T$. For each value of η, each function symbol f corresponds to a function $I_\eta(f)$ from $I_\eta(T_1) \times \ldots \times I_\eta(T_m)$ to $I_\eta(T)$, such that $I_\eta(f)(x_1, \ldots, x_m)$ is computable in polynomial time in the lengths of $x_1, \ldots, x_m$ and the value of η. Particular functions are predefined, and some of them use the infix notation: $M = N$ for the equality test, $M \neq N$ for the inequality test (both taking two values of the same type T and returning a value of type *bool*), $M \vee N$ for the boolean or, $M \wedge N$ for the boolean and, $\neg M$ for the boolean negation (taking and returning values of type *bool*).

In this calculus, terms represent computations on bitstrings. The replication index i is an integer which serves in distinguishing different copies of a replicated process $!^{i \leq n}$. (Replication indexes are typically used as array indexes.) The variable access $x[M_1, \ldots, M_m]$ returns the content of the cell of indexes $M_1, \dots, M_m$ of the array variable x. We use x, y, z, u as variable names. The function application $f(M_1, \dots, M_m)$ returns the result of applying function f to $M_1, \dots, M_m$.

The calculus distinguishes two kinds of processes: input processes Q are ready to receive a message on a channel; output processes P output a message on a channel after executing some internal computations. The input process 0 does nothing; $Q \mid Q'$ is the parallel composition of Q and Q'; $!^{i \leq n} Q$ represents n copies of Q in parallel, each with a different value of $i \in [1, n]$; *newChannel* $c; Q$ creates a new private channel c and executes Q; the semantics of the input $c[M_1, \dots, M_l](x_1[i_1, \dots, i_m] : T_1, \dots, x_k[i_1, \dots, i_m] : T_k); P$ will be explained below together with the semantics of the output.

The output process *new* $x[i_1, \dots, i_m] : T; P$ chooses a new random number uniformly in $I_\eta(T)$, stores it in $x[i_1, \dots, i_m]$, and executes P. (T must be a fixed-length type, because probabilistic polynomial-time Turing machines can choose random numbers uniformly only in such types.) Function symbols represent deterministic functions, so all random numbers must be chosen by *new* $x[i_1, \dots, i_m] : T$. Deterministic functions make automatic syntactic manipulations easier: we can duplicate a term without changing its value. The process *let* $x[i_1, \dots, i_m] : T = M$ *in* P stores the bitstring value of M (which must be in $I_\eta(T)$) in $x[i_1, \dots, i_m]$, and executes P. The process *if* M *then* P *else* P' executes P if M evaluates to 1 and P' if M evaluates to 0. Next, we explain the process *find* $(\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ *suchthat defined*$(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ *then* $P_j)$ *else* P, where $\widetilde{i}$ denotes a tuple $i_1, \dots, i_{m'}$. The order and array indexes on tuples are taken component-wise, so for instance, $u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ can be further abbreviated $\widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}$. A simple example is the following: *find* $u \leq n$ *suchthat defined*$(x[u]) \wedge x[u] = a$ *then* P' *else* P tries to find an index u such that $x[u]$ is defined and $x[u] = a$, and when such a u is found, it executes P' with that value of u; otherwise, it executes P. In other words, this *find* construct looks for the value a in the array x, and when a is found, it stores in u an index such that $x[u] = a$. Therefore, the *find* construct allows us to access arrays, which is key for our purpose. More generally, *find* $u_1[\widetilde{i}] \leq n_1, \dots, u_m[\widetilde{i}] \leq n_m$ *suchthat defined*$(M_1, \dots, M_l) \wedge M$ *then* P' *else* P tries to find values of $u_1, \dots, u_m$ for which $M_1, \dots, M_l$ are defined and M is true. In case of success, it executes P'. In case of failure, it executes P. This is further generalized to m branches: *find* $(\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ *suchthat defined*$(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ *then* $P_j)$ *else* P tries to find a branch j in $[1, m]$ such that there are values of $u_{j1}, \dots, u_{jm_j}$ for which $M_{j1}, \dots, M_{jl_j}$ are defined and M_j is true. In case of success, it executes P_j. In case of failure for all branches, it executes P. More formally, it evaluates the conditions *defined*$(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ for each j and each value of $u_{j1}[\widetilde{i}], \dots, u_{jm_j}[\widetilde{i}]$ in $[1, n_{j1}] \times \dots \times [1, n_{jm_j}]$. If none of these conditions is 1, it executes P. Otherwise, it chooses randomly with uniform[1] probability one j and one value of $u_{j1}[\widetilde{i}], \dots, u_{jm_j}[\widetilde{i}]$ such that the corresponding condition is 1, and executes P_j.

Finally, let us explain the output $\overline{c[M_1, \dots, M_l]}\langle N_1, \dots, N_k \rangle; Q$. A channel $c[M_1, \dots, M_l]$ consists of both a channel name c and a tuple of terms $M_1, \dots, M_l$. Channel names c allow us to define private channels to which the adversary can never have access, by *newChannel* c. (This is useful in the proofs, although all channels of protocols are often public.) Terms $M_1, \dots, M_l$ are intuitively analogous to IP addresses and ports which are numbers that the adversary may guess. Two channels are equal when they have the same channel name and terms that evaluate to the same bitstrings. A semantic configuration always consists of a single output process (the process currently being executed) and several input processes. When the output process executes $\overline{c[M_1, \dots, M_l]}\langle N_1, \dots, N_k \rangle; Q$, one looks for an input on the same channel and with the same arity in the available input processes. If no such input process is found, the process blocks. Otherwise, one such input process $c[M'_1, \dots, M'_l](x_1[i_1, \dots, i_m] : T_1, \dots,$

[1] A probabilistic polynomial-time Turing machine can choose a random number uniformly in a set of cardinal m only when m is a power of 2. When m is not a power of 2, we in fact use an approximate algorithm, as follows. We choose a random integer r uniformly among $[0, 2^k - 1]$ for a certain k large enough. When r is in $[0, (2^k \operatorname{div} m) \times m - 1]$, $r \bmod m$ returns a random integer in $[0, m - 1]$, with the same probability for all elements of $[0, m - 1]$. Otherwise, we can do anything, for example blocking. The probability of being in this case is smaller than $m/2^k$ so it can be made as small as we wish by choosing a large enough k.

$x_k[i_1, \ldots, i_m] : T_k)$; P is chosen randomly with uniform probability. The communication is then executed: for each $j \leq k$, the output message N_j is evaluated, its result is truncated to length $\mathrm{maxlen}_\eta(c)$, the obtained bitstring is stored in $x_j[i_1, \ldots, i_m]$ if it is in $I_\eta(T_j)$ (otherwise the process blocks). Finally, the output process P that follows the input is executed. The input process Q that follows the output is stored in the available input processes for future execution. Note that the syntax requires an output to be followed by an input process, as in [29]. If one needs to output several messages consecutively, one can simply insert fictitious inputs between the outputs. The adversary can then schedule the outputs by sending messages to these inputs. Also note that the internal computations of an output process are executed sequentially without any interruption (the other processes in parallel do not run until a communication is performed). This is important in our calculus to avoid race conditions, for example when several processes would look for an u such that $x[u]$ is defined and satisfies certain conditions, and when it is not found, define a certain $x[i]$.

Using different channels for each input and output allows the adversary to control the network. For instance, we may write $!^{i \leq n} c[i](x[i] : T) \ldots \overline{c'[i]}\langle M \rangle \ldots$ The adversary can then decide which copy of the replicated process receives its message, simply by sending it on $c[i]$ for the appropriate value of i.

We write *if* M *then* P as an abbreviation for *if* M *then* P *else* $\overline{\mathit{yield}}\langle\rangle$, and similarly for a *find* without *else* clause. ("*else* 0" would not be syntactically correct.) A trailing 0 after an output is omitted.

Variables can be defined by assignments, inputs, restrictions, and array lookups. The *current replication indexes* at a certain program point in a process are $i_1, \ldots, i_m$ where the replications above the considered program point are $!^{i_1 \leq n_1} \ldots !^{i_m \leq n_m}$. We often abbreviate $x[i_1, \ldots, i_m]$ by x when $i_1, \ldots, i_m$ are the current replication indexes, but it should be kept in mind that this is only an abbreviation. Variables defined under a replication must be arrays: for example $!^{i_1 \leq n_1} \ldots !^{i_m \leq n_m} \mathit{let}\ x[i_1, \ldots, i_m] : T = M\ \mathit{in}\ \ldots$ More formally, we require the following invariant:

Invariant 1 (Single definition). The process Q_0 satisfies Invariant 1 if and only if

1. in a definition of $x[i_1, \ldots, i_m]$ in Q_0, the indexes $i_1, \ldots, i_m$ of x are the current replication indexes at that definition, and
2. two different definitions of the same variable x in Q_0 are in different branches of a *if* or a *find*.

Invariant 1 guarantees that each variable is assigned at most once for each value of its indexes. (Indeed, item 2 shows that only one definition of each variable can be executed for given indexes in each trace).

Invariant 2 (Defined variables). The process Q_0 satisfies Invariant 2 if and only if every occurrence of a variable access $x[M_1, \ldots, M_m]$ in Q_0 is either

– syntactically under the definition of $x[M_1, \ldots, M_m]$ (in which case $M_1, \ldots, M_m$ are in fact the current replication indexes at the definition of x);
– or in a *defined* condition in a *find* process;
– or in M'_j or P_j in a process of the form *find* $(\bigoplus_{j=1}^{m''} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}$ *suchthat defined*$(M'_{j1}, \ldots, M'_{jl_j}) \wedge M'_j$ *then* $P_j)$ *else* P where for some $k \leq l_j$, $x[M_1, \ldots, M_m]$ is a subterm of M'_{jk}.

Invariant 2 guarantees that variables can be accessed only when they have been initialized. It checks that the definition of the variable access is either in scope (first item) or checked by a *find* (last item). Both invariants are checked by the prover for the initial game, and preserved by all game transformations.

We say that a function $f : T_1 \times \ldots \times T_m \rightarrow T$ is *poly-injective* when it is injective and its inverses can be computed in polynomial time, that is, there exist functions $f_j^{-1} : T \rightarrow T_j$ $(1 \leq j \leq m)$ such that $f_j^{-1}(f(x_1, \ldots, x_m)) = x_j$ and f_j^{-1} can be computed in polynomial time in the length of $f(x_1, \ldots, x_m)$ and in the security parameter. When f is poly-injective, we define a pattern matching construct *let* $f(x_1, \ldots, x_m) = M$ *in* P *else* Q as an abbreviation for *let* $y : T = M$ *in let* $x_1 : T_1 =$

$f_1^{-1}(y)$ *in* ... *let* $x_m : T_m = f_m^{-1}(y)$ *in if* $f(x_1, \ldots, x_m) = y$ *then* P *else* Q. We naturally generalize this construct to *let* $N = M$ *in* P *else* Q where N is built from poly-injective functions and variables.

Let us introduce two cryptographic primitives that we use in the following.

Definition 1. *Let T_{mr}, T_{mk}, and T_{ms} be types that correspond intuitively to random seeds, keys, and message authentication codes, respectively; T_{mr} is a fixed-length type. A message authentication code [14] consists of three function symbols:*

- *$mkgen : T_{mr} \to T_{mk}$ where $I_\eta(mkgen) = mkgen_\eta$ is the key generation algorithm taking as argument a random bitstring, and returning a key. (Usually, mkgen is a randomized algorithm; here, since we separate the choice of random numbers from computation, mkgen takes an additional argument representing the random coins.)*
- *$mac : bitstring \times T_{mk} \to T_{ms}$ where $I_\eta(mac) = mac_\eta$ is the mac algorithm taking as argument a message and a key, and returning the corresponding tag. (We assume here that mac is deterministic; we could easily encode a randomized mac by adding an additional argument as for mkgen.)*
- *$check : bitstring \times T_{mk} \times T_{ms} \to bool$ where $I_\eta(check) = check_\eta$ is a checking algorithm such that $check_\eta(m, k, t) = 1$ if and only if t is a valid mac of message m under key k. (Since mac is deterministic, $check_\eta(m, k, t)$ is typically $mac_\eta(m, k) = t$.)*

We have $\forall m \in Bitstring, \forall r \in I_\eta(T_{mr}), check_\eta(m, mkgen_\eta(r), mac_\eta(m, mkgen_\eta(r))) = 1$.

A mac is secure against existential forgery under chosen message attack if and only if for all polynomials q,

$$\max_{\mathcal{A}} \Pr[r \overset{R}{\leftarrow} I_\eta(T_{mr}); k \leftarrow mkgen_\eta(r); (m, t) \leftarrow \mathcal{A}^{mac_\eta(.,k), check_\eta(.,k,.)} : check_\eta(m, k, t)]$$

is negligible, where the adversary $\mathcal{A}$ is any probabilistic Turing machine, running in time $q(\eta)$, with oracle access to $mac_\eta(., k)$ and $check_\eta(., k, .)$, and $\mathcal{A}$ has not called $mac_\eta(., k)$ on message m.

Definition 2. *Let T_r and T_r' be fixed-length types; let T_k and T_e be types. A symmetric encryption scheme [12] (stream cipher) consists of three function symbols $kgen : T_r \to T_k$, $enc : bitstring \times T_k \times T_r' \to T_e$, and $dec : T_e \times T_k \to bitstring_\perp$, with $I_\eta(kgen) = kgen_\eta$, $I_\eta(enc) = enc_\eta$, $I_\eta(dec) = dec_\eta$, such that for all $m \in Bitstring$, $r \in I_\eta(T_r)$, and $r' \in I_\eta(T_r')$, $dec_\eta(enc_\eta(m, kgen_\eta(r), r'), kgen_\eta(r)) = m$.*

Let $LR(x, y, b) = x$ if $b = 0$ and $LR(x, y, b) = y$ if $b = 1$, defined only when x and y are bitstrings of the same length. A stream cipher is IND-CPA (satisfies indistinguishability under chosen plaintext attacks) if and only if for all polynomials q,

$$\max_{\mathcal{A}} 2\Pr[b \overset{R}{\leftarrow} \{0,1\}; r \overset{R}{\leftarrow} I_\eta(T_r); k \leftarrow kgen_\eta(r); b' \leftarrow \mathcal{A}^{r' \overset{R}{\leftarrow} I_\eta(T_r'); enc_\eta(LR(.,.,b),k,r')} : b' = b] - 1$$

is negligible, where the adversary $\mathcal{A}$ is any probabilistic Turing machine, running in time $q(\eta)$, with oracle access to the left-right encryption algorithm which given two bitstrings a_0 and a_1 of the same length, returns $r' \overset{R}{\leftarrow} I_\eta(T_r'); enc_\eta(LR(a_0, a_1, b), k, r')$, that is, encrypts a_0 when $b = 0$ and a_1 when $b = 1$.

Example 1. Let us consider the following trivial protocol:

$$A \to B : e, mac(e, x_{mk}) \quad \text{where } e = enc(x_k', x_k, x_r'') \text{ and } x_r'', x_k' \text{ are fresh random numbers}$$

A and B are assumed to share a key x_k for a stream cipher and a key x_{mk} for a message authentication code. A creates a fresh key x_k', and sends it encrypted under x_k to B. A mac is appended to the message, in order to guarantee integrity. The goal of the protocol is that x'_k should be a secret key shared between A and B. This protocol can be modeled in our calculus by the following process Q_0:

$$\begin{aligned}
Q_0 = {} & \mathit{start}(); \mathit{new}\ x_r : T_r; \mathit{let}\ x_k : T_k = \mathit{kgen}(x_r)\ \mathit{in} \\
& \quad \mathit{new}\ x'_r : T_{mr}; \mathit{let}\ x_{mk} : T_{mk} = \mathit{mkgen}(x'_r)\ \mathit{in}\ \overline{c}\langle\rangle; (Q_A \mid Q_B) \\
Q_A = {} & !^{i \leq n} c_A[i](); \mathit{new}\ x'_k : T_k; \mathit{new}\ x''_r : T'_r; \\
& \quad \mathit{let}\ x_m : \mathit{bitstring} = \mathit{enc}(\mathit{k2b}(x'_k), x_k, x''_r)\ \mathit{in}\ \overline{c_A[i]}\langle x_m, \mathit{mac}(x_m, x_{mk})\rangle \\
Q_B = {} & !^{i' \leq n} c_B[i'](x'_m, x_{ma}); \mathit{if}\ \ \mathit{check}(x'_m, x_{mk}, x_{ma})\ \mathit{then}\ \mathit{let}\ i_\perp(\mathit{k2b}(x''_k)) = \mathit{dec}(x'_m, x_k)\ \mathit{in}\ \overline{c_B[i']}\langle\rangle
\end{aligned}$$

When Q_0 receives a message on channel *start*, it begins execution: it generates the keys x_k and x_{mk} by choosing random coins x_r and $x_{r'}$ and applying the appropriate key generation algorithms. Then it yields control to the context (the adversary), by outputting on channel c. After this output, n copies of processes for A and B are ready to be executed, when the context outputs on channels $c_A[i]$ or $c_B[i]$ respectively. In a session that runs as expected, the context first sends a message on $c_A[i]$. Then Q_A creates a fresh key x'_k (T_k is assumed to be a fixed-length type), encrypts it under x_k with random coins x''_r, computes the *mac* of the encryption under x_{mk}, and sends the ciphertext and the mac on $c_A[i]$. The function $\mathit{k2b} : T_k \to \mathit{bitstring}$ is the natural injection $I_\eta(\mathit{k2b})(x) = x$; it is needed only for type conversion. The context is then expected to forward this message on $c_B[i]$. When Q_B receives this message, it checks the mac, decrypts, and stores the obtained key in x''_k. (The function $i_\perp : \mathit{bitstring} \to \mathit{bitstring}_\perp$ is the natural injection; it is useful to check that decryption succeeded.) This key x''_k should be secret.

The context is responsible for forwarding messages from A to B. It can send messages in unexpected ways in order to mount an attack.

This trivial running example is sufficient to illustrate the main features of our prover. Section 6 presents results obtained on more realistic protocols.

We denote by $\mathrm{var}(P)$ the set of variables that occur in P, and by $\mathrm{fc}(P)$ the set of free channels of P. (We use similar notations for input processes.)

2.2 Type System

We use a type system to check that bitstrings of the proper type are passed to each function, and that array indexes are used correctly.

To be able to type variable accesses used not under their definition (such accesses are guarded by a *find* construct), the type-checking algorithm proceeds in two passes. In the first pass, we build a type environment $\mathcal{E}$, which maps variable names x to types $T_1 \times \ldots \times T_m \to T$, where $T_1, \ldots, T_m$ are the interval types of the indexes of x, and T is the type of $x[i_1, \ldots, i_m]$. This type environment is built as follows:

– If x is defined by $\mathit{new}\ x[i_1, \ldots, i_m] : T$, $\mathit{let}\ x[i_1, \ldots, i_m] : T = M$, or $c[M_1, \ldots, M_l](\ldots, x[i_1, \ldots, i_m] : T, \ldots)$, and the replications above this subprocess are $!^{i_1 \leq n_1}, \ldots, !^{i_m \leq n_m}$, then $\mathcal{E}(x) = [1, n_1] \times \ldots \times [1, n_m] \to T$.
– If u is defined by $\mathit{find} \ldots \oplus \ldots u[i_1, \ldots, i_m] \leq n \ldots\ \ \mathit{suchthat}\ \mathit{defined}(\ldots) \wedge \ldots \mathit{then} \ldots \oplus \ldots$ and the replications above this *find* are $!^{i_1 \leq n_1}, \ldots, !^{i_m \leq n_m}$, then $\mathcal{E}(u) = [1, n_1] \times \ldots \times [1, n_m] \to [1, n]$.

We require that all definitions of the same variable x yield the same value of $\mathcal{E}(x)$, so that $\mathcal{E}$ is properly defined.

A process can then be typechecked in the type environment $\mathcal{E}$ using the rules of Figure 2. This figure defines three judgments:

– $\mathcal{E} \vdash M : T$ means that term M has type T in environment $\mathcal{E}$.

$$\frac{\mathcal{E}(i) = T}{\mathcal{E} \vdash i : T} \text{ (TIndex)} \qquad \frac{\mathcal{E}(x) = T_1 \times \ldots \times T_m \to T \qquad \forall j \leq m, \mathcal{E} \vdash M_j : T_j}{\mathcal{E} \vdash x[M_1, \ldots, M_m] : T} \text{ (TVar)}$$

$$\frac{f : T_1 \times \ldots \times T_m \to T \qquad \forall j \leq m, \mathcal{E} \vdash M_j : T_j}{\mathcal{E} \vdash f(M_1, \ldots, M_m) : T} \text{ (TFun)}$$

$$\mathcal{E} \vdash 0 \text{ (TNil)} \qquad \frac{\mathcal{E} \vdash Q \qquad \mathcal{E} \vdash Q'}{\mathcal{E} \vdash Q \mid Q'} \text{ (TPar)} \qquad \frac{\mathcal{E}[i \mapsto [1, n]] \vdash Q}{\mathcal{E} \vdash !^{i \leq n} Q} \text{ (TRepl)}$$

$$\frac{\mathcal{E} \vdash Q}{\mathcal{E} \vdash \mathit{newChannel}\ c; Q} \text{ (TNewChannel)}$$

$$\frac{\forall j \leq l, \mathcal{E} \vdash M_j : T'_j \qquad \forall j \leq k, \mathcal{E} \vdash x_j[\widetilde{i}] : T_j \qquad \mathcal{E} \vdash P}{\mathcal{E} \vdash c[M_1, \ldots, M_l](x_1[\widetilde{i}] : T_1, \ldots, x_k[\widetilde{i}] : T_k); P} \text{ (TIn)}$$

$$\frac{\forall j \leq l, \mathcal{E} \vdash M_j : T'_j \qquad \forall j \leq k, \mathcal{E} \vdash N_j : T_j \qquad \mathcal{E} \vdash Q}{\mathcal{E} \vdash \overline{c[M_1, \ldots, M_l]}\langle N_1, \ldots, N_k \rangle; Q} \text{ (TOut)}$$

$$\frac{T \text{ fixed-length type} \qquad \mathcal{E} \vdash x[\widetilde{i}] : T \qquad \mathcal{E} \vdash P}{\mathcal{E} \vdash \mathit{new}\ x[\widetilde{i}] : T; P} \text{ (TNew)}$$

$$\frac{\mathcal{E} \vdash M : T \qquad \mathcal{E} \vdash x[\widetilde{i}] : T \qquad \mathcal{E} \vdash P}{\mathcal{E} \vdash \mathit{let}\ x[\widetilde{i}] : T = M\ \mathit{in}\ P} \text{ (TLet)}$$

$$\frac{\mathcal{E} \vdash M : \mathit{bool} \qquad \mathcal{E} \vdash P \qquad \mathcal{E} \vdash P'}{\mathcal{E} \vdash \mathit{if}\ M\ \mathit{then}\ P\ \mathit{else}\ P'} \text{ (TIf)}$$

$$\frac{\begin{array}{c}\forall j \leq m, \forall k \leq m_j, \mathcal{E} \vdash u_{jk}[\widetilde{i}] : [1, n_{jk}] \\ \forall j \leq m, \forall k \leq l_j, \mathcal{E} \vdash M_{jk} : T_{jk} \qquad \forall j \leq m, \mathcal{E} \vdash M_j : \mathit{bool} \qquad \forall j \leq m, \mathcal{E} \vdash P_j \qquad \mathcal{E} \vdash P\end{array}}{\mathcal{E} \vdash \mathit{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathit{suchthat}\ \mathit{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathit{then}\ P_j)\ \mathit{else}\ P} \text{ (TFind)}$$

Fig. 2. Typing rules

– $\mathcal{E} \vdash P$ and $\mathcal{E} \vdash Q$ mean that the output process P and the input process Q are well-typed in environment $\mathcal{E}$, respectively.

In $x[M_1, \ldots, M_m]$, $M_1, \ldots, M_m$ must be of the suitable interval type. When $f(M_1, \ldots, M_m)$ is called, and $f : T_1 \times \ldots \times T_m \to T$, M_j must be of type T_j, and $f(M_1, \ldots, M_m)$ is then of type T. The type system requires each subterm to be well-typed. Furthermore, in $\mathit{let}\ x : T = M\ \mathit{in}\ P$, M must be of type T. In $\mathit{if}\ M\ \mathit{then}\ P_1\ \mathit{else}\ P_2$, M must be of type bool. Similarly, for

$$\mathit{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathit{suchthat}\ \mathit{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathit{then}\ P_j)\ \mathit{else}\ P$$

M_j is of type bool for all $j \leq m$. In $!^{i \leq n} Q$, i is of type $[1, n]$ in Q. For $\mathit{new}\ x[\widetilde{i}] : T$, T must be a fixed-length type.

Invariant 3 (Typing). The process Q_0 satisfies Invariant 3 if and only the type environment $\mathcal{E}$ for Q_0 is well-defined, and $\mathcal{E} \vdash Q_0$.

We require the adversary to be well-typed. This requirement does not restrict its computing power, because it can always define type-cast functions $f : T \to T'$ to bypass the type system. Similarly, the type system does not restrict the class of protocols that we consider, since the protocol may contain type-cast functions. The type system just makes explicit which set of bitstrings may appear at each point of the protocol.

We say that an occurrence of a term M in a process Q is of type T when $\mathcal{E} \vdash M : T$ where $\mathcal{E}$ is the type environment of Q extended with $i \mapsto [1,n]$ for each replication $!^{i \leq n}$ above M in Q.

2.3 Formal Semantics

The formal semantics is presented in Figure 3. A semantic configuration is a quadruple $E, P, \mathcal{Q}, \mathcal{C}$, where E is an environment mapping array cells to bitstrings or $\bot$, P is the output process currently scheduled, $\mathcal{Q}$ is the multiset of input processes running in parallel with P, $\mathcal{C}$ is the set of channels already created. The semantics is defined by reduction rules of the form $E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_{\eta,t} E', P', \mathcal{Q}', \mathcal{C}'$ meaning that $E, P, \mathcal{Q}, \mathcal{C}$ reduces to $E', P', \mathcal{Q}', \mathcal{C}'$ with probability p, when the security parameter is η. The value of the security parameter is often omitted to lighten the notation. The index t just serves in distinguishing reductions that yield the same configuration with the same probability in different ways, so that the probability of a certain reduction can be computed correctly:

$$\Pr[E, P, \mathcal{Q}, \mathcal{C} \rightarrow_\eta E', P', \mathcal{Q}', \mathcal{C}'] = \sum_{E,P,\mathcal{Q},\mathcal{C} \xrightarrow{p}_{\eta,t} E',P',\mathcal{Q}',\mathcal{C}'} p$$

The probability of a trace is computed as follows:

$$\Pr[E_1, P_1, \mathcal{Q}_1, \mathcal{C}_1 \rightarrow_\eta \ldots \rightarrow_\eta E'_m, P'_m, \mathcal{Q}'_m, \mathcal{C}'_m] = \prod_{j=1}^{m-1} \Pr[E_j, P_j, \mathcal{Q}_j, \mathcal{C}_j \rightarrow_\eta E'_{j+1}, P'_{j+1}, \mathcal{Q}'_{j+1}, \mathcal{C}'_{j+1}]$$

We define an auxiliary relation for evaluating terms: $E, M \Downarrow_\eta a$, or simply $E, M \Downarrow a$, means that the term M evaluates to the bitstring a in environment E. Rule (Cst) simply evaluates constants to themselves. This rule serves for replication indexes, which are substituted with constant values when reducing the replication. Rule (Var) looks for the value of the array variable in the environment. Rule (Fun) evaluates the function call. Rules (Def1) and (Def2) evaluate conditions of *find*: When some M_k is not defined, *defined*$(M_1, \ldots, M_l) \wedge M$ returns 0 (false) by (Def1). Otherwise, it returns the boolean value of M by (Def2).

We use an auxiliary reduction relation $\rightsquigarrow_\eta$, or simply $\rightsquigarrow$, for reducing input processes. This relation transforms configurations of the form $E, \mathcal{Q}, \mathcal{C}$. Rule (Nil) removes nil processes. Rules (Par) and (Repl) expand parallel compositions and replications, respectively. Rule (NewChannel) creates a new channel and adds it to $\mathcal{C}$. Semantic configurations are considered equivalent modulo renaming of channels in $\mathcal{C}$, so that a single semantic configuration is obtained after applying (NewChannel). Rule (Input) evaluates the terms in the input channel. The input itself is not executed: the communication is done by the (Output) rule. The relation $\rightsquigarrow$ is convergent (confluent and terminating), so it has normal forms. Since processes in $\mathcal{Q}$ in configurations $E, P, \mathcal{Q}, \mathcal{C}$ are in normal form by $\rightsquigarrow$, they always start with an input.

Rules (New) to (Find2) simply reduce the scheduled process. As explained in the footnote page 5, we use an approximately uniform probability distribution for choosing an element among a set S when $m = |S|$ is not a power of 2. Let k be the smallest integer such that $2^k \geq m$. We choose a random integer r uniformly among $[0, 2^{k+f(\eta)} - 1]$ for a certain function f. When r is in $[0, (2^{k+f(\eta)} \operatorname{div} m \times m) - 1]$, r mod m returns a random integer in $[0, m-1]$, with the same probability for all elements of $[0, m-1]$. When r is in $[2^{k+f(\eta)} \operatorname{div} m \times m, 2^{k+f(\eta)} - 1]$, we can do anything; we choose to block. The probability of being in this case is $(2^{k+f(\eta)} \mod m)/2^{k+f(\eta)} \leq m/2^{k+f(\eta)} \leq 1/2^{f(\eta)}$, so it can be made as small as we wish by choosing a large enough $f(\eta)$. We choose $f(\eta) \geq \alpha\eta$ for some $\alpha > 0$, so that it is negligible. The probability of choosing each element of S is then among$(S) = \frac{2^{k+f(\eta)} \operatorname{div} m}{2^{k+f(\eta)}}$. Then among$(S)$ approximates $1/m$. Rules (Find1) and (Find2) evaluate a *find*. They compute the value of all conditions $D_j \wedge M_j$ of this *find* for all possible values $\widetilde{v}$ of the indexes $\widetilde{u_j[\widetilde{a'}]}$. When all these conditions are false, rule (Find2) executes the *else* branch of the *find*. When at least one of these conditions is true, rule (Find1) chooses

Terms and *find* conditions:

$$E, a \Downarrow a \quad \text{(Cst)} \qquad \frac{\forall j \leq m, E, M_j \Downarrow a_j \qquad x[a_1, \dots, a_m] \in \mathrm{Dom}(E)}{E, x[M_1, \dots, M_m] \Downarrow E(x[a_1, \dots, a_m])} \quad \text{(Var)}$$

$$\frac{\forall j \leq m, E, M_j \Downarrow a_j \qquad f : T_1 \times \ldots \times T_m \to T \qquad \forall j \leq m, a_j \in I_\eta(T_j)}{E, f(M_1, \dots, M_m) \Downarrow I_\eta(f)(a_1, \dots, a_m)} \quad \text{(Fun)}$$

$$\frac{\neg \forall k \leq l, \exists a_k, E, M_k \Downarrow a_k}{E, \mathit{defined}(M_1, \dots, M_l) \wedge M \Downarrow 0} \quad \text{(Def1)}$$

$$\frac{\forall k \leq l, \exists a_k, E, M_k \Downarrow a_k \qquad E, M \Downarrow a \qquad a \in \{0, 1\}}{E, \mathit{defined}(M_1, \dots, M_l) \wedge M \Downarrow a} \quad \text{(Def2)}$$

Input processes:

$$E, \{0\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow E, \mathcal{Q}, \mathcal{C} \quad \text{(Nil)}$$

$$E, \{Q_1 \mid Q_2\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow E, \{Q_1, Q_2\} \uplus \mathcal{Q}, \mathcal{C} \quad \text{(Par)}$$

$$E, \{!^{i \leq n} Q\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow E, \{Q\{a/i\} \mid a \in [1, I_\eta(n)]\} \uplus \mathcal{Q}, \mathcal{C} \quad \text{(Repl)}$$

$$\frac{c' \notin \mathcal{C}}{E, \{\mathit{newChannel}\ c; Q\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow E, \{Q\{c'/c\}\} \uplus \mathcal{Q}, \mathcal{C} \cup \{c'\}} \quad \text{(NewChannel)}$$

$$\frac{\forall j \leq l, E, M_j \Downarrow a_j}{E, \{c[M_1, \dots, M_l](x_1[\widetilde{a'}] : T_1, \dots, x_k[\widetilde{a'}] : T_k); P\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow E, \{c[a_1, \dots, a_l](x_1[\widetilde{a'}] : T_1, \dots, x_k[\widetilde{a'}] : T_k); P\} \uplus \mathcal{Q}, \mathcal{C}} \quad \text{(Input)}$$

reduce$(E, \mathcal{Q}, \mathcal{C})$ is the normal form of $E, \mathcal{Q}, \mathcal{C}$ by $\rightsquigarrow$

Output processes:

$$\frac{T \text{ fixed-length type} \qquad a \in I_\eta(T)}{E, \mathit{new}\ x[\widetilde{a'}] : T; P, \mathcal{Q}, \mathcal{C} \xrightarrow{\frac{1}{|I_\eta(T)|}}_{N(a)} E[x[\widetilde{a'}] \mapsto a], P, \mathcal{Q}, \mathcal{C}} \quad \text{(New)}$$

$$\frac{E, M \Downarrow a \qquad a \in I_\eta(T)}{E, \mathit{let}\ x[\widetilde{a'}] : T = M\ \mathit{in}\ P, \mathcal{Q}, \mathcal{C} \xrightarrow{1}_{L} E[x[\widetilde{a'}] \mapsto a], P, \mathcal{Q}, \mathcal{C}} \quad \text{(Let)}$$

$$\frac{E, M \Downarrow 1}{E, \mathit{if}\ M\ \mathit{then}\ P_1\ \mathit{else}\ P_2, \mathcal{Q}, \mathcal{C} \xrightarrow{1}_{I1} E, P_1, \mathcal{Q}, \mathcal{C}} \quad \text{(If1)}$$

$$\frac{E, M \Downarrow 0}{E, \mathit{if}\ M\ \mathit{then}\ P_1\ \mathit{else}\ P_2, \mathcal{Q}, \mathcal{C} \xrightarrow{1}_{I2} E, P_2, \mathcal{Q}, \mathcal{C}} \quad \text{(If2)}$$

$$\frac{\begin{array}{c}\forall j \leq m, \forall \widetilde{v} \leq \widetilde{n_j}, E[\widetilde{u_j}[\widetilde{a'}] \mapsto \widetilde{v}], D_j \wedge M_j \Downarrow a_{j,\widetilde{v}} \\ S = \{j, \widetilde{v} \mid a_{j,\widetilde{v}} = 1\} \qquad a_{j_0,\widetilde{v_0}} = 1 \qquad E_{j_0,\widetilde{v_0}} = E[\widetilde{u_{j_0}}[\widetilde{a'}] \mapsto \widetilde{v_0}]\end{array}}{E, \mathit{find}\ (\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{a'}] \leq \widetilde{n_j}\ \mathit{suchthat}\ D_j \wedge M_j\ \mathit{then}\ P_j)\ \mathit{else}\ P, \mathcal{Q}, \mathcal{C} \xrightarrow{\text{among}(S)}_{F1(j_0,\widetilde{v_0})} E_{j_0,\widetilde{v_0}}, P_{j_0}, \mathcal{Q}, \mathcal{C}} \quad \text{(Find1)}$$

$$\frac{\forall j \leq m, \forall \widetilde{v} \leq \widetilde{n_j}, E[\widetilde{u_j}[\widetilde{a'}] \mapsto \widetilde{v}], D_j \wedge M_j \Downarrow 0}{E, \mathit{find}\ (\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{a'}] \leq \widetilde{n_j}\ \mathit{suchthat}\ D_j \wedge M_j\ \mathit{then}\ P_j)\ \mathit{else}\ P, \mathcal{Q}, \mathcal{C} \xrightarrow{1}_{F2} E, P, \mathcal{Q}, \mathcal{C}} \quad \text{(Find2)}$$

$$\frac{\begin{array}{c}\forall j \leq l, E, M_j \Downarrow a_j \qquad \forall j \leq k, E, N_j \Downarrow b_j \qquad E, \mathcal{Q}', \mathcal{C}' = \text{reduce}(E, \{Q''\}, \mathcal{C}) \\ S = \{Q \in \mathcal{Q} \mid Q = c[a_1, \dots, a_l](x'_1[\widetilde{a''}] : T'_1, \dots, x'_k[\widetilde{a''}] : T'_k).P' \text{ for some } x'_1, \dots, x'_k, \widetilde{a''}, T'_1, \dots, T'_k, P'\} \\ Q_0 = c[a_1, \dots, a_l](x_1[\widetilde{a'}] : T_1, \dots, x_k[\widetilde{a'}] : T_k).P \in S \qquad \forall j \leq k, b'_j = b_j \& (2^{\text{maxlen}_\eta(c)} - 1) \in I_\eta(T_j)\end{array}}{E, \overline{c[M_1, \dots, M_l]}\langle N_1, \dots, N_k \rangle.Q'', \mathcal{Q}, \mathcal{C} \xrightarrow{S(Q_0) \times \text{among}(S)}_{O(Q_0)} E[x_1[\widetilde{a'}] \mapsto b'_1, \dots, x_k[\widetilde{a'}] \mapsto b'_k], P, \mathcal{Q} \uplus \mathcal{Q}' \setminus \{Q_0\}, \mathcal{C}'} \quad \text{(Output)}$$

Fig. 3. Semantics

one such true case (for $j = j_0$ and $\widetilde{v} = \widetilde{v_0}$) with approximately uniform probability, and executes the corresponding *then* branch of the find.

Rule (Output) performs communications: it evaluates the terms in the channel and the sent messages, selects an input on the desired channel randomly, and immediately executes the communication. The scheduled process after this rule is the receiving process. (The process blocks if no suitable input is available.)

The initial configuration for running process Q_0 is $\mathrm{initConfig}(Q_0) = \emptyset, \overline{\mathit{start}}\langle\rangle, \mathcal{Q}, \mathcal{C}$ where $\emptyset, \mathcal{Q}, \mathcal{C} = \mathrm{reduce}(\emptyset, \{Q_0\}, \mathrm{fc}(Q_0))$.

We show the following properties:

P1. If Q_0 satisfies Invariant 1, then each variable is defined at most once for each value of its array indexes in a trace of Q_0.

P2. If Q_0 satisfies Invariant 2, then in traces of Q_0, the test $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$ in rule (Var) always succeeds, except when the considered term occurs in a *defined* condition of a *find*.

P3. If Q_0 satisfies Invariant 3, then in traces of Q_0, the tests T *fixed-length type* in rule (New), $a \in I_\eta(T)$ in rule (Let), $\forall j \leq m, a_j \in I_\eta(T_j)$ in rule (Fun), and the test $a \in \{0, 1\}$ in rule (Def2) always succeed.

P4. For each process Q, there exists a probabilistic polynomial time Turing machine that simulates Q. (Processes run in polynomial time since the number of processes created by a replication and the length of messages sent on channels are bounded by polynomials.) Conversely, our calculus can simulate a probabilistic polynomial-time Turing machine, simply by choosing coins by *new* and by applying a function symbol defined to perform the same computations as the Turing machine.

2.4 Observational Equivalence

A context is a process containing a hole $[\,]$. An evaluation context C is a context built from $[\,]$, *newChannel* $c; C$, $Q \mid C$, and $C \mid Q$. We use an evaluation context to represent the adversary. We denote by $C[Q]$ the process obtained by replacing the hole $[\,]$ in the context C with the process Q.

Definition 3. *Let c be a channel name and a be a bitstring. We say that $E, P, \mathcal{Q}, \mathcal{C}$* executes $\overline{c}\langle a\rangle$ immediately *when $P = \overline{c}\langle M\rangle.Q$ and $E, M \Downarrow a$ for some Q and M.*

The probability that Q executes $\overline{c}\langle a\rangle$ is denoted $\Pr[Q \rightsquigarrow_\eta \overline{c}\langle a\rangle]$. When $c \in \mathrm{fc}(Q)$, $\Pr[Q \rightsquigarrow_\eta \overline{c}\langle a\rangle] = \sum_{\mathcal{T} \in \mathbb{T}} \Pr[\mathcal{T}]$ where $\mathbb{T}$ is the set of traces $\mathrm{initConfig}(Q) \rightarrow_\eta \dots \rightarrow_\eta E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ such that $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ executes $\overline{c}\langle a\rangle$ immediately and for all $j < m$, $E_j, P_j, \mathcal{Q}_j, \mathcal{C}_j$ does not execute $\overline{c}\langle a\rangle$ immediately. When $c \notin \mathrm{fc}(Q)$, $\Pr[Q \rightsquigarrow_\eta \overline{c}\langle a\rangle] = 0$.

Definition 4 (Observational equivalence). *Let Q and Q' be two processes, and V a set of variables. Assume that Q and Q' satisfy invariants 1, 2, and 3 and the variables of V are defined in Q and Q', with the same types.*

An evaluation context is said to be acceptable *for Q, Q', V if and only if $\mathrm{var}(C) \cap (\mathrm{var}(Q) \cup \mathrm{var}(Q')) \subseteq V$ and $C[Q]$ satisfies Invariants 1, 2, and 3. (Then $C[Q']$ also satisfies these invariants.)*

We say that Q and Q' are observationally equivalent *with public variables V, written $Q \approx^V Q'$, when for all evaluation contexts C acceptable for Q, Q', V, for all channels c and bitstrings a, $|\Pr[C[Q] \rightsquigarrow_\eta \overline{c}\langle a\rangle] - \Pr[C[Q'] \rightsquigarrow_\eta \overline{c}\langle a\rangle]|$ is negligible.*

Our definition of observational equivalence is similar to that of [36]. Intuitively, the goal of the adversary represented by context C is to distinguish Q from Q'. When it succeeds, it performs a different output, for example $\overline{c}\langle 0\rangle$ when it has recognized Q and $\overline{c}\langle 1\rangle$ when it has recognized Q'. When $Q \approx^V Q'$, the context has negligible probability of distinguishing Q from Q'.

The unusual requirement on variables of C comes from the presence of arrays and of the associated *find* construct which gives C direct access to variables of Q and Q': the context C is allowed to access variables of Q and Q' only when they are in V. (In more standard settings, the calculus does not have constructs that allow the context to access variables of Q and Q'.) The following result is not difficult to prove:

Lemma 1. *$\approx^V$ is an equivalence relation, and $Q \approx^V Q'$ implies that $C[Q] \approx^{V'} C[Q']$ for all evaluation contexts C acceptable for Q, Q', V and all $V' \subseteq V \cup (\mathrm{var}(C) \setminus (\mathrm{var}(Q) \cup \mathrm{var}(Q')))$.*

We denote by $Q \approx_0^V Q'$ the particular case in which for all evaluation contexts C acceptable for Q, Q', V, for all channels c and bitstrings a, $\Pr[C[Q] \rightsquigarrow_\eta \overline{c}\langle a\rangle] = \Pr[C[Q'] \rightsquigarrow_\eta \overline{c}\langle a\rangle]$. When V is empty, we write $Q \approx Q'$ instead of $Q \approx^V Q'$ and $Q \approx_0 Q'$ instead of $Q \approx_0^V Q'$.

3 Game Transformations

In this section, we describe the game transformations that allow us to transform the process that represents the initial protocol into a process on which the desired security property can be proved directly, by criteria given in Section 4. These transformations are parametrized by the set V of variables that the context can access. As we shall see in Section 4, V contains variables that we would like to prove secret. These transformations transform a process Q_0 into a process Q'_0 such that $Q_0 \approx^V Q'_0$.

3.1 Syntactic Transformations

RemoveAssign(x): When x is defined by an assignment *let* $x[i_1, \dots, i_l] : T = M$ *in* P, we replace x with its value. Precisely, the transformation is performed only when x does not occur in M (non-cyclic assignment). When x has several definitions, we simply replace $x[i_1, \dots, i_l]$ with M in P. (For accesses to x guarded by *find*, we do not know which definition of x is actually used. In this case, applying the transformation **SArename**(x) defined below before **RemoveAssign**(x) may allow us to remove all assignments to x.) When x has a single definition, we replace everywhere in the game $x[M_1, \dots, M_l]$ with $M\{M_1/i_1, \dots, M_l/i_l\}$.

We additionally update the *defined* conditions of *find* to preserve Invariant 2, and to maintain the requirement that $x[M_1, \dots, M_l]$ is defined when it was required in the initial game. (Each defined condition *defined*$(N_1, \dots, N_m)$ that contains x is changed as follows: Let $N'_1, \dots, N'_{m'}$ be the subterms of $N_1, \dots, N_m$ of root x. Let $N''_1, \dots, N''_{m+m'}$ be obtained from $N_1, \dots, N_m$ and $N'_1, \dots, N'_{m'}$ by substituting $M\{M_1/i_1, \dots, M_l/i_l\}$ for $x[M_1, \dots, M_l]$ except at the root. The new defined condition contains $N''_1, \dots, N''_{m+m'}$, as well as for each $x[M_1, \dots, M_l]$ in $N''_1, \dots, N''_{m+m'}$, the array accesses that occur in $M\{M_1/i_1, \dots, M_l/i_l\}$.)

When $x \in V$, its definition is kept unchanged. Otherwise, when x is not referred to at all after the transformation, we remove the definition of x. When x is referred to only at the root of *defined* tests, we replace its definition with a constant. (The definition point of x is important, but not its value.)

Example 2. In the process of Example 1, **RemoveAssign**(x_{mk}) substitutes $\mathit{mkgen}(x'_r)$ for x_{mk} in the whole process and removes the assignment *let* $x_{mk} : T_{mk} = \mathit{mkgen}(x'_r)$. After this substitution, $\mathit{mac}(x_m, x_{mk})$ becomes $\mathit{mac}(x_m, \mathit{mkgen}(x'_r))$ and $\mathit{check}(x'_m, x_{mk}, x_{ma})$ becomes $\mathit{check}(x'_m, \mathit{mkgen}(x'_r), x_{ma})$, thus exhibiting terms required in Section 3.3. The situation is similar for **RemoveAssign**(k).

RemoveAssign(**useless**): As a particular case of the previous procedure, we remove useless assignments, that is, assignments to x when x is unused and assignments *let* $x[\widetilde{i}] : T = y[\widetilde{M}]$. Since removing such assignments may also remove uses of other variables, we repeat this removal until a fixpoint is reached.

SArename(x): The transformation **SArename** (single assignment rename) aims at renaming variables so that each variable has a single definition in the game; this is useful for distinguishing cases depending on which definition of x has set $x[\widetilde{i}]$. This transformation can be applied only when $x \notin V$. When x has $m > 1$ definitions, we rename each definition of x to a different variable $x_1, \dots, x_m$. Terms $x[\widetilde{i}]$ under a definition of $x_j[\widetilde{i}]$ are then replaced with $x_j[\widetilde{i}]$. Each branch of find $FB = \widetilde{u}[\widetilde{i}] \leq \widetilde{n}$ *suchthat defined*$(M'_1, \dots, M'_{l'}) \wedge M$ *then* P where $x[M_1, \dots, M_l]$ is a subterm of some M'_k for $k \leq l'$ is replaced with m branches $FB\{x_j[M_1, \dots, M_l]/x[M_1, \dots, M_l]\}$ for $1 \leq j \leq m$.

Example 3. Consider the following process

$$start(); new\ r_A : T_r; let\ k_A : T_k = kgen(r_A)\ in\ new\ r_B : T_r; let\ k_B : T_k = kgen(r_B)\ in\ \overline{yield}\langle\rangle; (Q_K \mid Q_S)$$

$$Q_K = !^{i \leq n} c[i](h : T_h, k : T_k) if\ h = A\ then\ let\ k' : T_k = k_A\ in\ \overline{yield}\langle\rangle\ else$$
$$if\ h = B\ then\ let\ k' : T_k = k_B\ in\ \overline{yield}\langle\rangle\ else\ let\ k' : T_k = k\ in\ \overline{yield}\langle\rangle$$

$$Q_S = !^{i' \leq n'} c'[i'](h' : T_h); find\ u \leq n\ suchthat\ defined(h[u], k'[u]) \wedge h' = h[u]\ then\ P_1(k'[u])\ else\ P_2$$

The process Q_K stores in (h, k') a table of pairs (host name, key): the key for A is k_A, for B k_B, and for any other h, the adversary can choose the key k. The process Q_S queries this table of keys to find the key $k'[u]$ of host h', then executes $P_1(k'[u])$. If h' is not found, it executes P_2.

By the transformation **SArename**(k'), we can perform a case analysis, to distinguish the cases in which $k' = k_A$, $k' = k_B$, or $k' = k$. After transformation, we obtain the following processes:

$$Q'_K = !^{i \leq n} c[i](h : T_h, k : T_k) if\ h = A\ then\ let\ k'_1 : T_k = k_A\ in\ \overline{yield}\langle\rangle\ else$$
$$if\ h = B\ then\ let\ k'_2 : T_k = k_B\ in\ \overline{yield}\langle\rangle\ else\ let\ k'_3 : T_k = k\ in\ \overline{yield}\langle\rangle$$

$$Q'_S = !^{i' \leq n'} c'[i'](h' : T_h); find\ u \leq n\ suchthat\ defined(h[u], k'_1[u]) \wedge h' = h[u]\ then\ P_1(k'_1[u])$$
$$\oplus\ u \leq n\ suchthat\ defined(h[u], k'_2[u]) \wedge h' = h[u]\ then\ P_1(k'_2[u])$$
$$\oplus\ u \leq n\ suchthat\ defined(h[u], k'_3[u]) \wedge h' = h[u]\ then\ P_1(k'_3[u])\ else\ P_2$$

After the simplification (described below), Q'_S becomes:

$$Q''_S = !^{i' \leq n'} c'[i'](h' : T_h); find\ u \leq n\ suchthat\ defined(h[u], k'_1[u]) \wedge h' = A\ then\ P_1(k_A)$$
$$\oplus\ u \leq n\ suchthat\ defined(h[u], k'_2[u]) \wedge h' = B\ then\ P_1(k_B)$$
$$\oplus\ u \leq n\ suchthat\ defined(h[u], k'_3[u]) \wedge h' = h[u]\ then\ P_1(k[u])\ else\ P_2$$

since, when $k'_1[u]$ is defined, $k'_1[u] = k_A$ and $h[u] = A$, and similarly for $k'_2[u]$ and $k'_3[u]$.

SArename(**auto**): As a particular case of the previous procedure, when x has $m > 1$ definitions and all variable accesses to x are of the form $x[i_1, \dots, i_l]$ under a definition of $x[i_1, \dots, i_l]$, where $i_1, \dots, i_l$ are the current replication indexes at this definition of x, we rename x to $x_1, \dots, x_m$ with a different name for each definition.

MoveNew We move restrictions downwards in the code as much as possible, when they have no array accesses. A *new* $x[\widetilde{i}] : T$ cannot be moved under a replication, or under a parallel composition when both sides use x, or a test *if* M *then* ... *else* ..., let *let* $y[\widetilde{i}] : T = M$ *in* ..., input $c[M_1, \dots, M_l](x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k)$, output $\overline{c[M_1, \dots, M_l]}\langle N_1, \dots, N_k\rangle$ when x occurs in $M, M_1, \dots, M_l, N_1, \dots, N_k$, or a *find* when the conditions use x. It can be moved under the other constructs, duplicating it if necessary, when we move it under a *if* or a *find* that uses x in several branches. Note that when the restriction *new* $x[\widetilde{i}] : T$ cannot be moved under an input, a parallel composition, or a replication, it must be written above the output that is located above the considered input, parallel composition or replication, so that the syntax of processes is not violated.

When this transformation duplicates a *new* $x[\widetilde{i}] : T$ by moving it under a *if* or a *find* that uses x in several branches, a subsequent **SArename**(x) enables us to distinguish several cases depending in which branch x is created, which is useful in some proofs.

Proposition 1. *Let Q_0 be a process that satisfies Invariants 1, 2, and 3, and Q'_0 the process obtained from Q_0 by one of the transformations above. Then Q'_0 satisfies Invariants 1, 2, and 3, and $Q_0 \approx^V Q'_0$.*

3.2 Simplification and Elimination of Collisions

In this section, we define the transformation **Simplify**, which is used to simplify games.

User-defined Rewrite Rules The user can give two kinds of information:

- claims of the form $\forall x_1 : T_1, \dots, \forall x_m : T_m, M$ which mean that for all environments E, if for all $j \leq m$, $E(x_j) \in I_\eta(T_j)$, then $E, M \Downarrow 1$.
  For example, considering mac and stream ciphers as in Definitions 1 and 2 respectively, we have:
  $$\forall r : T_{mr}, \forall m : \mathit{bitstring}, \mathit{check}(m, \mathit{mkgen}(r), \mathit{mac}(m, \mathit{mkgen}(r))) = 1 \qquad (\mathit{mac})$$
  $$\forall m : \mathit{bitstring}; \forall r : T_r, \forall r' : T'_r, \mathit{dec}(\mathit{enc}(m, \mathit{kgen}(r), r'), \mathit{kgen}(r)) = i_\perp(m) \qquad (\mathit{enc})$$
  We express the poly-injectivity of the function $k2b$ of Example 1 by
  $$\forall x : T_k, \forall y : T_k, (k2b(x) = k2b(y)) = (x = y) \qquad \forall x : T_k, k2b^{-1}(k2b(x)) = x \qquad (k2b)$$
  where $k2b^{-1}$ is a function symbol that denotes the inverse of $k2b$. We have similar formulas for $i_\perp$.
  Such claims must be well-typed, that is, $\{x_1 \mapsto T_1, \dots, x_m \mapsto T_m\} \vdash M : \mathit{bool}$.
  They are translated into rewrite rules as follows:
  - If M is of the form $M_1 = M_2$ and $\mathrm{var}(M_2) \subseteq \mathrm{var}(M_1)$, generate the rewrite rule $\forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \to M_2$.
  - If M is of the form $M_1 \neq M_2$, generate the rewrite rules $\forall x_1 : T_1, \dots, \forall x_m : T_m, (M_1 = M_2) \to 0$, $\forall x_1 : T_1, \dots, \forall x_m : T_m, (M_1 \neq M_2) \to 1$. (Such rules are used for instance to express that different constants are different.)
  - Otherwise, generate the rewrite rule $\forall x_1 : T_1, \dots, \forall x_m : T_m, M \to 1$.
- claims of the form $\mathit{new}\ y_1 : T'_1, \dots, \mathit{new}\ y_l : T'_l, \forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \approx M_2$ with $\mathrm{var}(M_2) \subseteq \mathrm{var}(M_1)$. Informally, these claims mean that M_1 and M_2 evaluate to the same bitstring except in cases of negligible probability, provided that $y_1, \dots, y_l$ are chosen randomly with uniform probability among $T'_1, \dots, T'_l$ respectively, and that $x_1, \dots, x_m$ are of type $T_1, \dots, T_m$. ($x_1, \dots, x_m$ may depend on $y_1, \dots, y_l$.) Formally, a first approach is to define these claims as
  $$\max_{\mathcal{A}} \Pr[E(y_1) \overset{R}{\leftarrow} I_\eta(T'_1); \dots E(y_l) \overset{R}{\leftarrow} I_\eta(T'_l); (E(x_1), \dots, E(x_m)) \leftarrow \mathcal{A}(E(y_1), \dots, E(y_l));$$
  $$E, M_1 \Downarrow a; E, M_2 \Downarrow a' : a \neq a'] \leq p(\eta)$$
  where $\mathcal{A}$ is a probabilistic Turing machine running in time $q(\eta)$, q is a polynomial, and $p(\eta)$ is negligible. However, this phrasing requires checking that the restrictions that create $y_1, \dots, y_l$ are pairwise distinct, which is sometimes delicate. (It may depend on the value of array indexes.) So we prefer the following definition, in which the substitution σ allows us to rename $y_1, \dots, y_l$ to possibly equal variables $y'_1, \dots, y'_{l'}$:
  The claim $\mathit{new}\ y_1 : T'_1, \dots, \mathit{new}\ y_l : T'_l, \forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \approx M_2$ means that for all substitutions σ that map $y_1, \dots, y_l$ to variables $y'_1, \dots, y'_{l'}$, such that $\sigma\{y_1, \dots, y_l\} = \{y'_1, \dots, y'_{l'}\}$ and for all $j \leq l$, if $\sigma y_j = y'_{j'}$ then $T''_{j'} = T'_j$, for all polynomials q,
  $$\max_{\mathcal{A}} \Pr[E(y'_1) \overset{R}{\leftarrow} I_\eta(T''_1); \dots E(y'_{l'}) \overset{R}{\leftarrow} I_\eta(T''_{l'}); (E(x_1), \dots, E(x_m)) \leftarrow \mathcal{A}(E(y'_1), \dots, E(y'_{l'}));$$
  $$E, \sigma M_1 \Downarrow a; E, \sigma M_2 \Downarrow a' : a \neq a'] \leq p(\eta)$$
  where $\mathcal{A}$ is a probabilistic Turing machine running in time $q(\eta)$, and $p(\eta)$ is negligible.

The claims need to be adapted to this definition. For instance, we write $\mathit{new}\ x : T; \mathit{new}\ y : T; \mathit{pkgen}(x) = \mathit{pkgen}(y) \approx x = y$ rather than $\mathit{new}\ x : T; \mathit{new}\ y : T; \mathit{pkgen}(x) = \mathit{pkgen}(y) \approx 0$, since we may have $\mathit{pkgen}(x) = \mathit{pkgen}(y)$ with probability 1 when x and y are in fact the same variable.
The above claim must be well-typed, that is, $\{x_1 \mapsto T_1, \dots, x_m \mapsto T_m, y_1 \mapsto T'_1, \dots, y_l \mapsto T'_l\} \vdash M_1 = M_2$.
This claim is translated into the rewrite rule $\mathit{new}\ y_1 : T'_1, \dots, \mathit{new}\ y_l : T'_l, \forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \rightarrow M_2$.

The term M reduces into M' by the rewrite rule $\mathit{new}\ y_1 : T'_1, \dots, \mathit{new}\ y_l : T'_l, \forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \rightarrow M_2$ if and only if $M = C[\sigma M_1]$, $M' = C[\sigma M_2]$, where C is a term context and σ is a substitution that maps x_j to any term of type T_j for all $j \leq m$, and y_j to terms to the form $x[\widetilde{M}]$ where x is defined by restrictions $\mathit{new}\ x : T'_j$ for all $j \leq l$.

The prover has built-in rewrite rules for defining boolean functions:

$$\neg 1 \rightarrow 0 \qquad \neg 0 \rightarrow 1 \qquad \forall x : \mathit{bool}, \neg(\neg x) \rightarrow x$$
$$\forall x : T, \forall y : T, \neg(x = y) \rightarrow x \neq y \qquad \forall x : T, \forall y : T, \neg(x \neq y) \rightarrow x = y$$
$$\forall x : T, x = x \rightarrow 1 \qquad \forall x : T, x \neq x \rightarrow 0$$
$$\forall x : \mathit{bool}, \forall y : \mathit{bool}, \neg(x \wedge y) \rightarrow (\neg x) \vee (\neg y) \qquad \forall x : \mathit{bool}, \forall y : \mathit{bool}, \neg(x \vee y) \rightarrow (\neg x) \wedge (\neg y)$$
$$\forall x : \mathit{bool}, x \wedge 1 \rightarrow x \qquad \forall x : \mathit{bool}, x \wedge 0 \rightarrow 0 \qquad \forall x : \mathit{bool}, x \vee 1 \rightarrow 1 \qquad \forall x : \mathit{bool}, x \vee 0 \rightarrow x$$

The prover also has support for commutative function symbols, that is, binary function symbols $f : T \times T \rightarrow T'$ such that for all $x, y \in I_\eta(T)$, $I_\eta(f)(x, y) = I_\eta(f)(y, x)$. For such symbols, all equality and matching tests are performed modulo commutativity. The functions $\wedge$, $\vee$, $=$, and $\neq$ are commutative. So, for instance, the last four rewrite rules above may also be used to rewrite $1 \wedge M$ into M, $0 \wedge M$ into 0, $1 \vee M$ into 1, and $0 \vee M$ into M. Used-defined functions may also be declared commutative; *xor* is an example of such a commutative function.

Dependency Analysis We say that M *characterizes* y intuitively when $\alpha M = M$ implies $(\alpha y)[\widetilde{M'}] = y[\widetilde{M}]$ for some $\widetilde{M}$ and $\widetilde{M'}$, where the renaming α maps each variable of M to a fresh variable, $y[\widetilde{M}]$ is a subterm of M, and $(\alpha y)[\widetilde{M'}]$ is a subterm of αM.

We use a simple rewriting prover to determine that. We consider the set of terms $\mathcal{M}_0 = \{\alpha M = M\}$, and we rewrite elements of $\mathcal{M}_0$ using the first kind of user-defined rewrite rules mentioned above and the rule $\{M_1 \wedge M_2\} \cup \mathcal{M}' \rightarrow \{M_1, M_2\} \cup \mathcal{M}'$.

When $\mathcal{M}_0$ can be rewritten to a set that contains $y[\widetilde{M}] = (\alpha y)[\widetilde{M'}]$ or $(\alpha y)[\widetilde{M'}] = y[\widetilde{M}]$ for some $\widetilde{M}$ and $\widetilde{M'}$, we have that M characterizes y.

We say that only_dep$(x) = S$ when intuitively, only variables in S depend on x, and the adversary cannot see the value of x. Formally, only_dep$(x) = S$ when S is the smallest set of variables containing x such that

– $S \cap V = \emptyset$.
– Variables of S do not occur in input or output channels or messages, that is, they do not occur in $M_1, \dots, M_m, N_1, \dots, N_k$, $x_1, \dots, x_k$ in the input $c[M_1, \dots, M_m](x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k)$ or in the output $\overline{c[M_1, \dots, M_m]}\langle N_1, \dots, N_k\rangle$.
– If a variable $y \in S$ occurs in M in $\mathit{let}\ x : T = M\ \mathit{in}\ P$, then M characterizes y and $x \in S$.
– Variables in S may occur in *defined* conditions of *find* but only at the root of them.
– All terms M in processes $\mathit{if}\ M\ \mathit{then}\ P_1\ \mathit{else}\ P_2$ and all terms M_j in processes $\mathit{find}\ (\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}\ \mathit{suchthat}\ \mathit{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathit{then}\ P_j)\ \mathit{else}\ P'$ are combinations by $\wedge$, $\vee$, or $\neg$ of terms that either do not contain variables in S or are of the form $M_1 = M_2$ or $M_1 \neq M_2$ where there exists

$y \in S$ such that M_1 characterizes y and no variable of S occurs in M_2, or the symmetric obtained by swapping M_1 and M_2.

The last item implies that the result of tests does not depend on the values of variables in S, except in cases of negligible probability. Indeed, the tests $M_1 = M_2$ with M_1 characterizes $y \in S$ and M_2 does not depend on variables in S are false except in cases of negligible probability. Similarly, the tests $M_1 \neq M_2$ are true except in cases of negligible probability.

The set S, when it exists, is computed by a fixpoint iteration, starting from $\{x\}$ and adding variables defined by assignments that depend on variables already in S.

Collecting True Facts from a Game We use *facts* to represent properties that hold at certain program points in processes. We consider two kinds of facts: *defined*(M) means that M is defined, and a term M means that M is true (the boolean term M evaluates to 1). The function collectFacts determines which facts hold at each program point of the game. More precisely, it computes a mapping TrueFacts from each occurrence P of a subprocess of the game, to the set of facts that hold at that occurrence. (It is important that P is an occurrence and not a process: processes at several occurrences may be equal, and must be distinguished from one another here.) The function collectFacts also computes a set $\mathrm{TrueFacts}_{\mathrm{def}}$ containing pairs $(x[\widetilde{i}], \mathcal{F})$ where $\mathcal{F}$ is a set of facts that hold if $x[\widetilde{i}]$ has been defined by a certain definition. (If there are several definitions of x, there is one such pair for each definition of x.)

The function collectFacts is defined in Figure 4. It is initially called with an empty set of facts: $\mathrm{collectFacts}(Q_0, \emptyset)$. It takes into account that $x[\widetilde{i}]$ may be defined by an input, a restriction, a let, or a find. Furthermore, when we execute *let* $x[\widetilde{i}] : T = M$ *in* P', $x[\widetilde{i}] = M$ holds in P' and holds when x is defined by that definition. When we execute *if* M *then* P_1 *else* P_2, M holds in P_1 and $\neg M$ holds in P_2. When we execute *find* $(\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ *suchthat defined*$(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ *then* $P_j)$ *else* P', M_j holds in P_j, $M_{j1}, \dots, M_{jl_j}, u_{j1}[\widetilde{i}], \dots, u_{jm_j}[\widetilde{i}]$ are defined in P_j, and these facts hold when x is defined by that definition.

After calling $\mathrm{collectFacts}(Q_0, \emptyset)$, we complete the computed sets $\mathrm{TrueFacts}_P$ by adding facts that we can deduce from facts *defined*(M). Precisely, if *defined*$(M) \in \mathrm{TrueFacts}_P$, and $x[M_1, \dots, M_m]$ is a subterm of M, we take into account facts that are known to be true at the definitions of x by adding them to $\mathrm{TrueFacts}_P$ as follows:

$$\mathrm{TrueFacts}_P \leftarrow \mathrm{TrueFacts}_P \cup (\cap_{(x[i_1,\dots,i_m],\mathcal{F}) \in \mathrm{TrueFacts}_{\mathrm{def}}} \mathcal{F}\{M_1/i_1, \dots, M_m/i_m\})$$

This operation may add new *defined* facts to $\mathrm{TrueFacts}_P$, so it is executed until a fixpoint is reached, except that, in order to avoid infinite loops, we do not execute this step for definitions *defined*(M) in which M contains nested occurrences of the same symbol (such as $x[\dots x[\dots] \dots]$).

Equational Prover We use an algorithm inspired by the Knuth-Bendix completion algorithm [26], with differences detailed below.

The prover manipulates pairs $\mathcal{F}, \mathcal{R}$ where $\mathcal{F}$ is a set of facts (M or *defined*(M)) and $\mathcal{R}$ is a set of rewrite rules $M_1 \to M_2$. We say that M reduces into M' by $M_1 \to M_2$ when $M = C[M_1]$ and $M' = C[M_2]$ for some term context C. (That is, all variables in rewrite rules of $\mathcal{R}$ are considered as constants.) The prover starts with a certain set of facts $\mathcal{F}$ and $\mathcal{R} = \emptyset$. Then the prover transforms the pairs $(\mathcal{F}, \mathcal{R})$ by the following rules (the rule $\frac{\mathcal{F},\mathcal{R}}{\mathcal{F}',\mathcal{R}'}$ means that $\mathcal{F}, \mathcal{R}$ is transformed into $\mathcal{F}', \mathcal{R}'$):

$$\frac{\mathcal{F} \cup \{F\}, \mathcal{R}}{\mathcal{F} \cup \{F'\}, \mathcal{R}} \text{ if } F \text{ reduces into } F' \text{ by a rule of } \mathcal{R} \text{ or a user-defined rewrite rule} \tag{1}$$

$$\frac{\mathcal{F} \cup \{M_1 \wedge M_2\}, \mathcal{R}}{\mathcal{F} \cup \{M_1, M_2\}, \mathcal{R}} \tag{2}$$

```
collectFacts$(Q, \mathcal{F})$ =
    TrueFacts$_Q = \mathcal{F}$
    if $Q = Q_1 \mid Q_2$ then collectFacts$(Q_1, \mathcal{F})$; collectFacts$(Q_2, \mathcal{F})$
    if $Q = !^{i \leq n} Q'$ then collectFacts$(Q', \mathcal{F})$
    if $Q = \mathit{newChannel}\ c; Q'$ then collectFacts$(Q', \mathcal{F})$
    if $Q = c[M_1, \ldots, M_l](x_1[\widetilde{i}] : T_1, \ldots, x_k[\widetilde{i}] : T_k); P$ then
        TrueFacts$_{\mathrm{def}}$ = TrueFacts$_{\mathrm{def}} \cup \{(x_j[\widetilde{i}], \mathcal{F}) \mid j \leq k\}$;
        collectFacts$(P, \mathcal{F} \cup \{\mathit{defined}(x_j[\widetilde{i}]) \mid j \leq k\})$

collectFacts$(P, \mathcal{F})$ =
    TrueFacts$_P = \mathcal{F}$
    if $P = \overline{c[M_1, \ldots, M_l]}\langle N_1, \ldots, N_k \rangle; Q$ then collectFacts$(Q, \mathcal{F})$
    if $P = \mathit{new}\ x[\widetilde{i}] : T; P'$ then
        TrueFacts$_{\mathrm{def}}$ = TrueFacts$_{\mathrm{def}} \cup \{(x[\widetilde{i}], \mathcal{F})\}$; collectFacts$(P', \mathcal{F} \cup \{\mathit{defined}(x[\widetilde{i}])\})$
    if $P = \mathit{let}\ x[\widetilde{i}] : T = M\ \mathit{in}\ P'$ then
        $\mathcal{F}' = \mathcal{F} \cup \{\mathit{defined}(x[\widetilde{i}]), x[\widetilde{i}] = M\}$; TrueFacts$_{\mathrm{def}}$ = TrueFacts$_{\mathrm{def}} \cup \{(x[\widetilde{i}], \mathcal{F}')\}$; collectFacts$(P', \mathcal{F}')$
    if $P = \mathit{if}\ M\ \mathit{then}\ P_1\ \mathit{else}\ P_2$ then
        collectFacts$(P_1, \mathcal{F} \cup \{M\})$; collectFacts$(P_2, \mathcal{F} \cup \{\neg M\})$
    if $P = \mathit{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathit{suchthat}\ \mathit{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathit{then}\ P_j)\ \mathit{else}\ P'$
    then
        for each $j \leq m$,
            $\mathcal{F}' = \mathcal{F} \cup \{\mathit{defined}(u_{j1}[\widetilde{i'}]), \ldots, \mathit{defined}(u_{jm_j}[\widetilde{i'}]), \mathit{defined}(M_{j1}), \ldots, \mathit{defined}(M_{jl_j}), M_j\}$
            TrueFacts$_{\mathrm{def}}$ = TrueFacts$_{\mathrm{def}} \cup \{(u_{j1}[\widetilde{i'}], \mathcal{F}'), \ldots, u_{jm_j}[\widetilde{i'}], \mathcal{F}')\}$
            collectFacts$(P_j, \mathcal{F}')$
        collectFacts$(P', \mathcal{F})$
```

Fig. 4. The function collectFacts

$$\frac{\mathcal{F} \cup \{x[M_1, \ldots, M_m] = x[M'_1, \ldots, M'_m]\}, \mathcal{R}}{\mathcal{F} \cup \{M_1 = M'_1, \ldots, M_m = M'_m\}, \mathcal{R}} \begin{array}{l}\text{when } x \text{ is defined by restrictions } \mathit{new}\ x : T \\ \text{and } T \text{ is a large type}\end{array} \tag{3}$$

$$\frac{\mathcal{F} \cup \{M_1 = M_2\}, \mathcal{R}}{\{0\}, \mathcal{R}} \begin{array}{l}\text{when } x \text{ occurs in } M_1,\ x \text{ is defined by restrictions } \mathit{new}\ x : T,\ T \text{ is a large type,} \\ M_1 \text{ characterizes } x, \text{ and either } M_2 \text{ is obtained by optionally applying function} \\ \text{symbols to terms of the form } y[\widetilde{M}] \text{ where } y \text{ is defined by restrictions and } y \neq x, \\ \text{or only\_dep}(x) = S \text{ and no variable of } S \text{ occurs in } M_2.\end{array} \tag{4}$$

$$\frac{\mathcal{F} \cup \{M = M'\}, \mathcal{R}}{\mathcal{F}, \mathcal{R} \cup \{M \rightarrow M'\}} \text{ if } M > M' \tag{5}$$

$$\frac{\mathcal{F}, \mathcal{R} \cup \{M_1 \rightarrow M_2\}}{\mathcal{F} \cup \{M_1 = M'_2\}, \mathcal{R}} \text{ if } M_2 \text{ reduces into } M'_2 \text{ by a rule of } \mathcal{R} \text{ or a user-defined rewrite rule} \tag{6}$$

$$\frac{\mathcal{F}, \mathcal{R} \cup \{M_1 \rightarrow M_2\}}{\mathcal{F} \cup \{M'_1 = M_2\}, \mathcal{R}} \text{ if } M_1 \text{ reduces into } M'_1 \text{ by a rule of } \mathcal{R} \tag{7}$$

We also use the symmetrics of Rules (4) and (5) obtained by swapping both sides of the equality.

Rule (1) simplifies facts using rewrite rules. Rule (2) decomposes conjunctions of facts. Rules (3) and (4) exploit the elimination of collisions between random values. Rule (3) takes into account that, when x is defined by a restriction of a large type, two different cells of x have a negligible probability of containing the same value. So when two cells of x contain the same value, we can conclude up to negligible probability that they are the same cell. Rule (4) expresses that M_1 and M_2 have a negligible probability of being equal when x is defined by a restriction of a large type, M_1 characterizes x, and M_2 does not depend of x.

Rule (5) is applied only when Rules (1) to (4) cannot be applied. Rule (5) transforms equations into rewrite rules by orienting them. We say that $M > M'$ when either M is the form $x[\widetilde{M}]$, x does not occur in M', and x is not defined by a restriction, or $M = x[M_1, \dots, M_m]$, $M' = x[M'_1, \dots, M'_m]$, and for all $j \leq m$, $M_j > M'_j$. Intuitively, our goal is to replace M with M' when M' defines the content of the variable M. (Notice that this is not an ordering; the Knuth-Bendix algorithm normally uses a reduction ordering to orient equations. However, we tried some reduction orderings, namely the lexicographic path ordering and the Knuth-Bendix ordering, and obtained disappointing results: the prover fails to prove many equalities because too many equations are left unoriented. The simple heuristic given above succeeds more often, at the expense of a greater risk of non-termination, but that does not cause problems in practice on our examples. We believe that this comes from the particular structure of equations, which come from *let* definitions and from tests, and tend to define variables from other variables without creating dependency cycles.)

Rules (6) and (7) are systematically applied to simplify all rewrite rules of $\mathcal{R}$ after a new rewrite rule has been added by Rule (5). Since all terms in rewrite rules of $\mathcal{R}$ are considered as constants, Rule (7) in fact includes the deduction of equations from critical pairs done by the standard Knuth-Bendix completion algorithm.

We say that $\mathcal{F}$ *yields a contradiction* when the prover starting from $(\mathcal{F}, \emptyset)$ derives 0.

Game Simplification

- Each term M in the game is replaced with a simplified term M' obtained by reducing M by user-defined rewrite rules (first point of this section) and the rewrite rules obtained from $\mathrm{TrueFacts}_{P_M}$ by the above equational prover where P_M is the smallest process containing M. The replacement is performed only when at least one user-defined rewrite rule has been used. (To avoid complicating the game by substituting all variables with their value.)
- If $P = \mathit{if}\ M\ \mathit{then}\ P_1\ \mathit{else}\ P_2$, and $\mathrm{TrueFacts}_{P_2} = \{\neg M\} \cup \mathrm{TrueFacts}_P$ yields a contradiction, then P is replaced with P_1. (The probability that P_2 is executed is negligible.)
- If $P = \mathit{if}\ M\ \mathit{then}\ P_1\ \mathit{else}\ P_2$, and $\mathrm{TrueFacts}_{P_1} = \{M\} \cup \mathrm{TrueFacts}_P$ yields a contradiction, then P is replaced with P_2.
- If $P = \mathit{find}\ (\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}\ \mathit{suchthat}\ \mathit{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathit{then}\ P_j)\ \mathit{else}\ P'$ and $\mathrm{TrueFacts}_{P_j}$ yields a contradiction, then the j-th branch of the *find* is removed.
- A *find* with no branches: *find else* P' is replaced with P'.
- If $P = \mathit{find}\ (\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}\ \mathit{suchthat}\ \mathit{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathit{then}\ P_j)\ \mathit{else}\ P'$ and for some j, for some $\widetilde{N}$, $\{\mathit{defined}(M_{j1}\{\widetilde{N}/\widetilde{u_j}\}), \dots, \mathit{defined}(M_{jl_j}\{\widetilde{N}/\widetilde{u_j}\})\} \subseteq \mathrm{TrueFacts}_P$ and $\{\neg M_j\{\widetilde{N}/\widetilde{u_j}\}\} \cup \mathrm{TrueFacts}_P$ yields a contradiction, then P' is replaced with $\overline{\mathit{yield}}\langle\rangle$. (The probability that P' is executed is negligible. The terms $\widetilde{N}$ are found by exploring the set of *defined* terms in $\mathrm{TrueFacts}_P$.)
- The *defined* conditions of *find* are updated so that Invariant 2 is satisfied. (When such a condition guarantees that M is defined, *defined*(M) implies *defined*(M'), and after simplification M' appears in the scope of this condition, then M' has to be added to this condition if it is not already present.)
- If $P = \mathit{find}\ (\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}\ \mathit{suchthat}\ \mathit{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathit{then}\ \overline{\mathit{yield}}\langle\rangle)\ \mathit{else}\ \overline{\mathit{yield}}\langle\rangle$ and the variables in $\widetilde{u_j}$ are not used outside P and are not in V, then P is replaced with $\overline{\mathit{yield}}\langle\rangle$.

$$
\begin{aligned}
&[\![(G_1,\dots,G_m)]\!] = [\![G_1]\!]^1 \mid \dots \mid [\![G_m]\!]^m\\
&[\![!^{i\le n} \mathit{new}\ y_1:T_1;\dots;\mathit{new}\ y_l:T_l;(G_1,\dots,G_m)]\!]^{\widetilde{j}}_{\widetilde{i}} =\\
&\quad !^{i\le n} c_{\widetilde{j}}[\widetilde{i},i]();\mathit{new}\ y_1:T_1;\dots;\mathit{new}\ y_l:T_l;\overline{c_{\widetilde{j}}[\widetilde{i},i]}\langle\rangle;([\![G_1]\!]^{\widetilde{j},1}_{\widetilde{i},i}\mid\dots\mid[\![G_m]\!]^{\widetilde{j},m}_{\widetilde{i},i})\\
&[\![(x_1:T_1,\dots,x_l:T_l)\to FP]\!]^{\widetilde{j}}_{\widetilde{i}} = c_{\widetilde{j}}[\widetilde{i}](x_1:T_1,\dots,x_l:T_l);[\![FP]\!]^{\widetilde{j}}_{\widetilde{i}}\\
&[\![M]\!]^{\widetilde{j}}_{\widetilde{i}} = \overline{c_{\widetilde{j}}[\widetilde{i}]}\langle M\rangle\\
&[\![\mathit{new}\ x[\widetilde{i}]:T;FP]\!]^{\widetilde{j}}_{\widetilde{i}} = \mathit{new}\ x[\widetilde{i}]:T;[\![FP]\!]^{\widetilde{j}}_{\widetilde{i}}\\
&[\![\mathit{let}\ x[\widetilde{i}]:T=M\ \mathit{in}\ FP]\!]^{\widetilde{j}}_{\widetilde{i}} = \mathit{let}\ x[\widetilde{i}]:T=M\ \mathit{in}\ [\![FP]\!]^{\widetilde{j}}_{\widetilde{i}}\\
&[\![\mathit{if}\ M\ \mathit{then}\ FP_1\ \mathit{else}\ FP_2]\!]^{\widetilde{j}}_{\widetilde{i}} = \mathit{if}\ M\ \mathit{then}\ [\![FP_1]\!]^{\widetilde{j}}_{\widetilde{i}}\ \mathit{else}\ [\![FP_2]\!]^{\widetilde{j}}_{\widetilde{i}}\\
&[\![\mathit{find}\ (\bigoplus_{j=1}^{m}\widetilde{u_j}[\widetilde{i}]\le\widetilde{n_j}\ \mathit{suchthat}\ \mathit{defined}(M_{j1},\dots,M_{jl_j})\wedge M_j\ \mathit{then}\ FP_j)\ \mathit{else}\ FP]\!]^{\widetilde{j}}_{\widetilde{i}} =\\
&\quad \mathit{find}\ (\bigoplus_{j=1}^{m}\widetilde{u_j}[\widetilde{i}]\le\widetilde{n_j}\ \mathit{suchthat}\ \mathit{defined}(M_{j1},\dots,M_{jl_j})\wedge M_j\ \mathit{then}\ [\![FP_j]\!]^{\widetilde{j}}_{\widetilde{i}})\ \mathit{else}\ [\![FP]\!]^{\widetilde{j}}_{\widetilde{i}}
\end{aligned}
$$

where $c_{\widetilde{j}}$ are pairwise distinct channels, $\widetilde{i} = i_1,\dots,i_{l'}$, and $\widetilde{j} = j_0,\dots,j_{l'}$.

Fig. 5. Translation from functional processes to processes

– If $P = \mathit{new}\ x:T;P'$ or $\mathit{let}\ x:T=M\ \mathit{in}\ P'$ and x is not used in the game and is not in V, then P is replaced with P'.
– If $P = \mathit{if}\ M\ \mathit{then}\ \overline{\mathit{yield}}\langle\rangle\ \mathit{else}\ \overline{\mathit{yield}}\langle\rangle$, then P is replaced with $\overline{\mathit{yield}}\langle\rangle$.

The following proposition shows the soundness of simplification. It is proved in Appendix B.3.

Proposition 2. *Let Q_0 be a process that satisfies Invariants 1, 2, and 3, and Q'_0 the corresponding process after simplification. Then Q'_0 also satisfies Invariants 1, 2, and 3, and $Q_0 \approx^V Q'_0$.*

3.3 Applying the Definition of Security of Primitives

The security of cryptographic primitives is defined using observational equivalences given as axioms. Importantly, this formalism allows us to specify many different primitives in a generic way. Such equivalences are then used by the prover in order to transform a game into another, observationally equivalent game, as explained in the following of this section.

The primitives are specified using equivalences of the form $(G_1,\dots,G_m)\approx(G'_1,\dots,G'_m)$ where G is defined by the following grammar:

$G ::=$	group of functions
$\quad !^{i\le n}\mathit{new}\ y_1:T_1;\dots;\mathit{new}\ y_l:T_l;(G_1,\dots,G_m)$	replication and restrictions ($l\ge 0, m\ge 1$)
$\quad (x_1:T_1,\dots,x_l:T_l)\to FP$	function ($l \ge 0$)

$FP ::=$	functional processes
$\quad M$	term
$\quad \mathit{new}\ x[\widetilde{i}]:T;FP$	random number generation (uniform)
$\quad \mathit{let}\ x[\widetilde{i}]:T=M\ \mathit{in}\ FP$	assignment
$\quad \mathit{if}\ M\ \mathit{then}\ FP_1\ \mathit{else}\ FP_2$	test
$\quad \mathit{find}\ (\bigoplus_{j=1}^{m}\widetilde{u_j}[\widetilde{i}]\le\widetilde{n_j}\ \mathit{suchthat}\ \mathit{defined}(M_{j1},\dots,M_{jl_j})\wedge M_j\ \mathit{then}\ FP_j)\ \mathit{else}\ FP$	array lookup

Intuitively, $(x_1:T_1,\dots,x_l:T_l)\to FP$ represents a function that takes as argument values $x_1,\dots,x_l$ of types $T_1,\dots,T_l$ respectively, and returns a result computed by FP. The observational equivalence

$(G_1,\dots,G_m) \approx (G'_1,\dots,G'_m)$ expresses that the adversary has a negligible probability of distinguishing functions in the left-hand side from corresponding functions in the right-hand side. Formally, functions can be encoded as processes that input their arguments and output their result on a channel, as shown in Figure 5. The translation of $!^{i\le n} \mathit{new}\ y_1 : T_1;\dots;\mathit{new}\ y_l : T_l;(G_1,\dots,G_m)$ inputs and outputs on channel $c_{\tilde{j}}$ so that the context can trigger the generation of random numbers $y_1,\dots,y_l$. The translation of $(x_1 : T_1,\dots,x_l : T_l) \rightarrow FP$ inputs the arguments of the function on channel $c_{\tilde{j}}$ and translates FP, which outputs the result of FP on $c_{\tilde{j}}$. (In the left-hand side, the result FP of functions must simply be a term M.) The observational equivalence $(G_1,\dots,G_m) \approx (G'_1,\dots,G'_m)$ is then an abbreviation for $[\![(G_1,\dots,G_m)]\!] \approx [\![(G'_1,\dots,G'_m)]\!]$.

For example, the security of a mac (Definition 1) is represented by the equivalence $L \approx R$ where:

$$\begin{aligned}
L = {} & !^{i''\le n''} \mathit{new}\ r : T_{mr}; (!^{i\le n}(x : \mathit{bitstring}) \rightarrow \mathit{mac}(x, \mathit{mkgen}(r)),\\
& \qquad !^{i'\le n'}(m : \mathit{bitstring}, ma : T_{ms}) \rightarrow \mathit{check}(m, \mathit{mkgen}(r), ma))\\
R = {} & !^{i''\le n''} \mathit{new}\ r : T_{mr}; (!^{i\le n}(x : \mathit{bitstring}) \rightarrow \mathit{mac}'(x, \mathit{mkgen}'(r)),\\
& \qquad !^{i'\le n'}(m : \mathit{bitstring}, ma : T_{ms}) \rightarrow \mathit{find}\ u \le n\ \mathit{suchthat}\ \mathit{defined}(x[u])\wedge\\
& \qquad\qquad (m = x[u]) \wedge \mathit{check}'(m, \mathit{mkgen}'(r), ma)\ \mathit{then}\ 1\ \mathit{else}\ 0)
\end{aligned} \qquad (\mathit{mac}_{\mathrm{eq}})$$

where mac', check', and mkgen' are function symbols with the same types as mac, check, and mkgen respectively. (We use different function symbols on the left- and right-hand sides, just to prevent a repeated application of the transformation induced by this equivalence. Since we add these function symbols, we also add the equation

$$\forall r : T_{mr}, \forall m : \mathit{bitstring}, \mathit{check}'(m, \mathit{mkgen}'(r), \mathit{mac}'(m, \mathit{mkgen}'(r))) = 1 \qquad (\mathit{mac}')$$

which restates Equation (mac) for mac', check', and mkgen'.) Intuitively, the equivalence $L \approx R$ leaves mac computations unchanged (except for the use of primed function symbols in R), and allows one to replace a mac checking $\mathit{check}(m, \mathit{mkgen}(r), ma)$ with a lookup in the array x of messages whose mac has been computed with key $\mathit{mkgen}(r)$: if m is found in the array x and $\mathit{check}(m, \mathit{mkgen}(r), ma)$, we return 1; otherwise, the check fails (up to negligible probability), so we return 0. (If the check succeeded with m not in the array x, the adversary would have forged a mac.) Obviously, the form of L requires that r is used only to compute or check macs, for the equivalence to be correct. Formally, the following result shows the correctness of our modeling. It is a fairly easy consequence of Definition 1.

Proposition 3. *Assuming* $(\mathit{mkgen}, \mathit{mac}, c)$ *is a message authentication code secure against existential forgery under chosen message attack,* $I_\eta(\mathit{mkgen}') = I_\eta(\mathit{mkgen})$, $I_\eta(\mathit{mac}') = I_\eta(\mathit{mac})$, *and* $I_\eta(\mathit{check}') = I_\eta(\mathit{check})$, *then* $[\![L]\!] \approx [\![R]\!]$.

Similarly, we represent the security of a IND-CPA stream cipher (Definition 2) by the equivalence:

$$\begin{aligned}
& !^{i'\le n'} \mathit{new}\ r : T_r; !^{i\le n}(x : \mathit{bitstring}) \rightarrow \mathit{new}\ r' : T'_r; \mathit{enc}(x, \mathit{kgen}(r), r')\\
\approx {} & !^{i'\le n'} \mathit{new}\ r : T_r; !^{i\le n}(x : \mathit{bitstring}) \rightarrow \mathit{new}\ r' : T'_r; \mathit{enc}'(Z(x), \mathit{kgen}'(r), r')
\end{aligned} \qquad (\mathit{enc}_{\mathrm{eq}})$$

where enc' and kgen' are function symbols with the same types as enc and kgen respectively, and $Z : \mathit{bitstring} \rightarrow \mathit{bitstring}$ is the function that returns a bitstring of the same length as its argument, consisting only of zeroes. Using equations such as $\forall x : T, Z(\mathit{T2b}(x)) = Z_T$, we can prove that $Z(\mathit{T2b}(x))$ does not depend on x when x is of a fixed-length type and $\mathit{T2b} : T \rightarrow \mathit{bitstring}$ is the natural injection. The representation of other primitives can be found in Appendix A.

We require the following conditions for the equivalences $L \approx R$ that model cryptographic primitives:

H0. $[\![L]\!]$ and $[\![R]\!]$ satisfy Invariants 1, 2, and 3. Furthermore, the result of each function in R has the same type as the result of the corresponding function of L.

H1. In L, the functional processes FP are simply terms M; all their array accesses use the current replication indexes. (Allowing *let* or *find* in L is difficult, because we need to recognize the terms M in a context and in a possibly syntactically modified form.)

H2. L and R have the same structure: same replications, same number of functions, same number of arguments with the same types for each function.

H3. The variables y_j defined by *new* and x_j defined by function inputs in L and R are distinct from other variables defined in R.

H4. Under $!^{i\leq n}$ with no restriction in L, one can have only a single function $(x_1 : T_1, \dots, x_l : T_l) \rightarrow FP$. (One can transform $!^{i\leq n}((\widetilde{x_1} : \widetilde{T_1}) \rightarrow FP_1, \dots, (\widetilde{x_m} : \widetilde{T_m}) \rightarrow FP_m, !^{i_1\leq n_1} \dots, \dots, !^{i_{m'}\leq n_{m'}} \dots)$ into $(!^{i\leq n}(\widetilde{x_1} : \widetilde{T_1}) \rightarrow FP_1, \dots, !^{i\leq n}(\widetilde{x_m} : \widetilde{T_m}) \rightarrow FP_m, !^{i_1\leq n'_1} \dots, \dots, !^{i_{m'}\leq n'_{m'}} \dots)$ in order to eliminate situations that do not satisfy this requirement.)

H5. Replications in L (resp. R) must have pairwise distinct bounds n. (This strengthens the typing: the typing then guarantees that, when several variables are accessed with the same array indexes, then these variables are defined under the same replication.)

H6. For all restrictions *new* $y : T$ that occur above a term M in L, y occurs in M. (This guarantees that, in Hypothesis H′3.1 below, $z_{jk}[M_{j1}, \dots, M_{jq_j}]$ is defined for all $j \leq l$ and $k \leq m_j$. With hypothesis H4, this guarantees that index_j is well-defined in Hypothesis H′3.1 below.)

H7. Finds in R are of the form

$$\mathit{find}\ (\bigoplus_{j=1}^{m} \widetilde{u_j} \leq \widetilde{n_j}\ \mathit{suchthat}\ \mathit{defined}(z_{j1}[\widetilde{u_{j1}}], \dots, z_{jl_j}[\widetilde{u_{jl_j}}]) \wedge M_j\ \mathit{then}\ FP_j)\ \mathit{else}\ FP'$$

where $\widetilde{u_{jk}}$ is the concatenation of a prefix of the current replication indexes (the same prefix for all k) and a non-empty prefix of $\widetilde{u_j}$, and at least one $\widetilde{u_{jk}}$ for $1 \leq k \leq l_j$ is the concatenation of a prefix of the current replication indexes with the whole sequence $\widetilde{u_j}$. Furthermore, there must exist $k \in \{1, \dots, l_j\}$ such that for all $k' \neq k$, $z_{jk'}$ is defined syntactically above all definitions of z_{jk} and $\widetilde{u_{jk'}}$ is a prefix of $\widetilde{u_{jk}}$. (This implies that the same find cannot access variables defined in different functions under the same replication in R.) Finally, variables z_{jk} must not be defined by a *find* in R. (Otherwise, the transformation would be considerably more complicated.)

Such equivalences $L \approx R$ are used by the prover by replacing a process Q_0 observationally equivalent to $C[[\![L]\!]]$ with a process Q'_0 observationally equivalent to $C[[\![R]\!]]$, for some evaluation context C. We now give sufficient conditions for a process to be equivalent to $C[[\![L]\!]]$. These conditions essentially guarantee that all uses of certain secret variables of Q_0, in a set S, can be implemented by calling functions of L.

We first define the function extract used in order to extract information from the left- or right-hand sides of the equivalence.

$$\begin{aligned}
&\mathrm{extract}((x_1 : T_1, \dots, x_l : T_l) \rightarrow M, ()) = (x_1 : T_1, \dots, x_l : T_l) \rightarrow M\\
&\mathrm{extract}(!^{i\leq n}\mathit{new}\ y_1 : T_1; \dots; \mathit{new}\ y_l : T_l; (G_1, \dots, G_m), (j_1, \dots, j_k)) =\\
&\qquad (y_1 : T_1, \dots, y_l : T_l), \mathrm{extract}(G_{j_1}, (j_2, \dots, j_k))\\
&\mathrm{extract}((G_1, \dots, G_m), (j_0, \dots, j_k)) = \mathrm{extract}(G_{j_1}, (j_1, \dots, j_k))
\end{aligned}$$

We rename the variables of Q_0 such that variables of L and R do not occur in Q_0. Assume that there exist a set of variables S and a set $\mathcal{M}$ of occurrences of terms in Q_0 such that:

H′1. $S \cap V = \emptyset$.

H′2. No term in $\mathcal{M}$ occurs in the condition part of a *find* $(\mathit{defined}(M_1, \dots, M_l) \wedge M)$ or in the channel of an input.

H′3. For each $M \in \mathcal{M}$, there exist a sequence $BL(M) = (j_0, \dots, j_l)$ such that $\mathrm{extract}(L, BL(M)) = (y_{11} : T_{11}, \dots, y_{1m_1} : T_{1m_1}), \dots, (y_{l1} : T_{l1}, \dots, y_{lm_l} : T_{lm_l}), (x_1 : T_1, \dots, x_m : T_m) \to N$ and a substitution σ such that $M = \sigma N$ (σ applies to the abbreviated form of N in which we write x instead of $x[\widetilde{i}]$) and

H′3.1. for all $j \leq l$ and $k \leq m_j$, σy_{jk} is a variable access $z_{jk}[M_{j1}, \dots, M_{jq_j}]$, with $z_{jk} \in S$. We define $z_{jk} = \mathrm{varImL}(y_{jk}, M)$. All definitions of z_{jk} in Q_0 are of the form *new* $z_{jk}[\dots] : T_{jk}$, and for all $k \leq m_j$, they occur under the same replications (but they may occur under different replications for different values of j). The sequence of array indexes $M_{j1}, \dots, M_{jq_j}$ is the same for all $k \leq m_j$ (but may depend on j). We denote by $\mathrm{index}_j(M)$ a substitution that maps the current replication indexes at the definition of z_{jk} to $M_{j1}, \dots, M_{jq_j}$ respectively. If $m_l = 0$, $\mathrm{index}_l(M)$ is not set by the previous definition, so we set $\mathrm{index}_l(M)$ to map the current replication indexes at M to themselves. When $j \neq j'$ or $k \neq k'$, $z_{jk} \neq z_{j'k'}$. For each $j < l$, there exists a substitution $\rho_j(M)$ such that $\mathrm{index}_j(M) = \mathrm{index}_{j+1}(M) \circ \rho_j(M)$ and the image of $\rho_j(M)$ does not contain the current replication indexes at M. We denote by $\mathrm{im\ index}_j(M)$ the sequence image by $\mathrm{index}_j(M)$ of the sequence of current replication indexes at the definition of z_{jk} (so, $\mathrm{im\ index}_j(M) = (M_{j1}, \dots, M_{jq_j})$). We define $\mathrm{im}\ \rho_j(M)$ similarly.

H′3.2. for all $j \leq m$, σx_j is a term of type T_j.

H′3.3. all occurrences in Q_0 of a variable in S are either as z_{jk} above or at the root of an argument of a *defined* test in a *find* process.

To make it precise which term M each element refers to, we add M as a subscript, writing $y_{jk,M}$ for y_{jk}, $z_{jk,M}$ for z_{jk}, $T_{jk,M}$ for T_{jk}, $x_{j,M}$ for x_j, $T_{j,M}$ for T_j, N_M for N, and σ_M for σ. We also define $\mathrm{nNew}_{j,M} = m_j$, $\mathrm{nNewSeq}_M = l$, and $\mathrm{nInput}_M = m$.

H′4. We say that two terms $M, M' \in \mathcal{M}$ share the first l' sequences of random variables when $y_{jk,M} = y_{jk,M'}$ and $z_{jk,M} = z_{jk,M'}$ for all $j \leq l'$ and $k \leq \mathrm{nNew}_{j,M} = \mathrm{nNew}_{j,M'} \neq 0$. Let l' be the greatest integer such that M and M' share the first l' sequences of random variables. Then

H′4.1. the sets of variables $\{z_{jk,M} \mid j > l' \text{ and } k \leq \mathrm{nNew}_{j,M}\}$ and $\{z_{jk,M'} \mid j > l' \text{ and } k \leq \mathrm{nNew}_{j,M'}\}$ must be disjoint.

H′4.2. $\rho_j(M) = \rho_j(M')$ for all $j < l'$.

H′4.3. if $l' = \mathrm{nNewSeq}_M$ and $N_M = N_{M'}$, then there exists M_0 such that $M = (\mathrm{index}_{l'}(M))M_0$, $M' = (\mathrm{index}_{l'}(M'))M_0$, and M_0 does not contain the current replication indexes at M or M'.

Then there exists a context C such that $Q_0 \approx_0^V C[\llbracket L \rrbracket]$.

Terms in $\mathcal{M}$ must not occur in conditions of *find* (Hypothesis H′2) because such terms may refer to variables defined by *find*, and by the transformation, these variables might be moved outside their scope, thus violating Invariant 2. Terms in $\mathcal{M}$ must not occur in the channel of an input, because after the transformation, the input process might need to perform computations by *find* or *let*, forbidden by the syntax. (This requirement is not a limitation in practice, since terms in channels of inputs are typically the current replication indexes, so they do not contain cryptographic primitives.)

In Hypothesis H′3, the sequence $BL(M)$ indicates which branch of L corresponds to the term M.

Hypothesis H′3.2 checks that the values received by inputs in L are of the proper type. Hypothesis H′3.1 checks that variables $z_{jk,M}$ that correspond to variables defined by *new* in L are of the proper type. The variables y_{jk} defined by *new* in L are used only in terms N in L. Correspondingly, Hypothesis H′3.3 checks that the corresponding variables $z_{jk,M} \in S$ are not used elsewhere in Q_0 and Hypothesis H′1 checks that they cannot be used directly by the context.

In L, for distinct j, k, the variables y_{jk} correspond to independent random numbers. Correspondingly, Hypothesis H′3.1 requires that the variables $z_{jk,M}$ are created by different restrictions for distinct j, k. In L, the variables y_{jk} are accessed with the same indexes for any k (but a fixed j). Correspondingly, Hypothesis H′3.1 requires that the variables $z_{jk,M}$ are accessed with the same indexes $\mathrm{im\ index}_j(M)$ for any k. When instances of N and N' both refer to y_{jk} with the same indexes, then they also refer to $y_{j'k'}$ with the same indexes when $j' \leq j$. Correspondingly, if M and M' refer to the same z_{jk}, by

Hypothesis H$'$4.1, they also refer to the same $z_{j'k'}$ for $j' \leq j$. Moreover, if $\mathrm{index}_j(M)$ and $\mathrm{index}_j(M')$ evaluate to the same bitstrings, then $\mathrm{index}_{j'}(M)$ and $\mathrm{index}_{j'}(M')$ also evaluate to the same bitstrings, since $\mathrm{index}_{j'}(M) = \mathrm{index}_j(M) \circ \rho_{j-1}(M) \circ \ldots \circ \rho_{j'}(M)$ by Hypothesis H$'$3.1 and $\rho_k(M) = \rho_k(M')$ for $k < j$ by Hypothesis H$'$4.2.

Finally, a term N in L is evaluated at most once for each value of the indexes of $y_{l1}, \ldots, y_{lm_l}$, so N is computed for a single value of the arguments $x_1, \ldots, x_m$. Correspondingly, by Hypothesis H$'$4.3, when M and M' share the $l = \mathrm{nNewSeq}_M$ sequences of random variables and $\mathrm{index}_l(M)$ and $\mathrm{index}_l(M')$ evaluate to the same bitstring, then M and M' evaluate to the same bitstring.

These conditions guarantee that we can establish a correspondence from the array cells of variables of S in Q_0 to the array cells of variables defined by *new* in L, and that this correspondence is an injective function; moreover, they also establish a correspondence between the terms $\sigma_M x$ where is a function argument in L, and the cells of x. More precisely, let $\widetilde{i}$ and $\widetilde{i'}$ be the sequences of current replication indexes at N_M in L and at M in Q_0, respectively. There exists a function mapIdx_M that maps the array indexes at M in Q_0 to the array indexes at N_M in L: the evaluation of M when $\widetilde{i'} = \widetilde{a}$ will correspond in $C[\llbracket L \rrbracket]$ to the evaluation of N_M when $\widetilde{i} = \mathit{mapIdx}_M(\widetilde{a})$. Thus, σ_M and mapIdx_M induce a correspondence between Q_0 and L: for all $M \in \mathcal{M}$, for all $x[\widetilde{i''}]$ that occur in N_M, $(\sigma_M x)\{\widetilde{a}/\widetilde{i'}\}$ corresponds to $x[\widetilde{i''}]\{\mathit{mapIdx}_M(\widetilde{a})/\widetilde{i}\}$, that is, $(\sigma_M x)\{\widetilde{a}/\widetilde{i'}\}$ in a trace of Q_0 has the same value as $x[\widetilde{i''}]\{\mathit{mapIdx}_M(\widetilde{a})/\widetilde{i}\}$ in the corresponding trace of $C[\llbracket L \rrbracket]$ ($\widetilde{i''}$ is a prefix of $\widetilde{i}$).

For example, consider a process Q_0 that contains $M_1 = \mathit{enc}(M'_1, \mathit{kgen}(x_r), x'_r[i_1])$ and $M_2 = \mathit{enc}(M'_2, \mathit{kgen}(x_r), x''_r[i_2])$ with $i_1 \leq n_1$, $i_2 \leq n_2$, and x_r, x'_r, x''_r bound by restrictions. Let $S = \{x_r, x'_r, x''_r\}$, $\mathcal{M} = \{M_1, M_2\}$, and $N_{M_1} = N_{M_2} = \mathit{enc}(x[i', i], \mathit{kgen}(r[i']), r'[i', i])$. The functions mapIdx_{M_1} and mapIdx_{M_2} are defined by $\mathit{mapIdx}_{M_1}(a_1) = (1, a_1)$ for $a_1 \in [1, I_\eta(n_1)]$ and $\mathit{mapIdx}_{M_2}(a_2) = (1, a_2 + I_\eta(n_1))$ for $a_2 \in [1, I_\eta(n_2)]$. Then $M'_1\{a_1/i_1\}$ corresponds to $x[1, a_1]$, x_r to $r[1]$, $x'_r[a_1]$ to $r'[1, a_1]$, $M'_2\{a_2/i_2\}$ to $x[1, a_2 + I_\eta(n_1)]$, and $x''_r[a_2]$ to $r'[1, a_2 + I_\eta(n_1)]$. The functions mapIdx_{M_1} and mapIdx_{M_2} are such that $x_{r'}[a_1]$ and $x_{r''}[a_2]$ never correspond to the same cell of r'; indeed, $x_{r'}[a_1]$ and $x_{r''}[a_2]$ are independent random numbers in Q_0, so their images in $C[\llbracket L \rrbracket]$ must also be independent random numbers.

The above correspondence must satisfy the following soundness conditions: when x is a function argument in L, the term that corresponds to $x[\widetilde{a'}]$ must have the same type as $x[\widetilde{a'}]$, and when two terms correspond to the same $x[\widetilde{a'}]$, they must evaluate to the same value (Hypothesis H$'$4.3); when x is bound by *new* $x : T$ in L, the term that corresponds to $x[\widetilde{a'}]$ must be evaluate to $z[\widetilde{a''}]$ where $z \in S$ and z is bound by *new* $z : T$ in Q_0, and the relation that associates $z[\widetilde{a''}]$ to $x[\widetilde{a'}]$ is an injective function. (It is easy to check that, in the previous example, these conditions are satisfied.)

We now describe how we construct a process Q'_0 such that $Q'_0 \approx_0^V C[\llbracket R \rrbracket]$.

– We first transform the right-hand side of the equivalence, R, as follows: for each $j_1, \ldots, j_l$, if $\mathrm{extract}(L, (j_1, \ldots, j_l)) = (y_{11} : T_{11}, \ldots, y_{1m_1} : T_{1m_1}), \ldots, (y_{l1} : T_{l1}, \ldots, y_{lm_l} : T_{lm_l}), (x_1 : T_1, \ldots, x_m : T_m) \rightarrow N$ with $m_l \neq 0$ and $\mathrm{extract}(R, (j_1, \ldots, j_l)) = (y'_{11} : T'_{11}, \ldots, y'_{1m'_1} : T'_{1m'_1}), \ldots, (y'_{l1} : T'_{l1}, \ldots, y'_{lm'_l} : T'_{lm'_l}), (x_1 : T_1, \ldots, x_m : T_m) \rightarrow FP$, for each *new* $z : T$ in FP,
  - we add z in the sequence of random variables $y'_{l1} : T'_{l1}, \ldots, y'_{lm'_l} : T'_{lm'_l}$;
  - if z does not occur in *defined* conditions of *find* in R, we remove *new* $z : T$ from FP;
  - otherwise, we replace *new* $z : T$ with *let* $z' : T = \mathit{cst}$ for some constant cst and add $z'[\widetilde{M}]$ to each *defined* condition of R that contains $z[\widetilde{M}]$.

  (In the right-hand side, a new random number must be chosen exactly for each different call to the function $(x_1 : T_1, \ldots, x_m : T_m) \rightarrow FP$. This would not be guaranteed without that transformation, because when the left-hand side N is evaluated at several occurrences with the same random numbers $y_{l1} : T_{l1}, \ldots, y_{lm_l} : T_{lm_l}$ ($m_l \neq 0$), these occurrences all correspond to a single call to $(x_1 : T_1, \ldots, x_m : T_m) \rightarrow N$, so a single call to $(x_1 : T_1, \ldots, x_m : T_m) \rightarrow FP$, but we create a copy of FP for each occurrence. After the transformation, FP contains no choice of random numbers, so we can evaluate it several times without changing the result. When $m_l = 0$, evaluations of N at several occurrences can correspond to different calls to $(x_1 : T_1, \dots, x_m : T_m) \to N$, so the transformation is not necessary.)

– For each $M \in \mathcal{M}$, let $\mathrm{extract}(R, BL(M)) = (y'_{11,M} : T'_{11,M}, \dots, y'_{1m'_1,M} : T'_{1m'_1,M}), \dots, (y'_{l1,M} : T'_{l1,M}, \dots, y'_{lm'_l,M} : T'_{lm'_l,M}), (x_{1,M} : T_{1,M}, \dots, x_{m,M} : T_{m,M}) \to FP_M$ with $l = \mathrm{nNewSeq}_M$, $m = \mathrm{nInput}_M$ and we define $\mathrm{nNew}'_{j,M} = m'_j$. We create fresh variables $z'_{jk,M} = \mathrm{varImR}(y'_{jk,M}, M)$ for each $j \leq \mathrm{nNewSeq}_M$, $k \leq \mathrm{nNew}'_{j,M}$, and $M \in \mathcal{M}$, such that if M and M' share the first l' sequences of random variables, then $z'_{jk,M} = z'_{jk,M'}$ for $j \leq l'$ and $k \leq \mathrm{nNew}'_{j,M}$. All variables $z'_{jk,M}$ are otherwise pairwise distinct.
We also create a fresh variable $\mathrm{varImR}(x_{j,M}, M)$ for each $j \leq \mathrm{nInput}_M$ and each $M \in \mathcal{M}$, and a fresh variable $\mathrm{varImR}(z, M)$ for each variable z defined by *let* or *new* in FP_M and each $M \in \mathcal{M}$.

– If a *defined* condition of a *find* contains $z_{j1,M}[M_1, \dots, M_{l'}]$ for some M, we add *defined*$(z'_{jk',M}[M_1, \dots, M_{l'}])$ for all $k' \leq \mathrm{nNew}'_{j,M}$ to this condition. (So that accesses to $z'_{jk',M}[M_1, \dots, M_{l'}]$ created when transforming term M satisfy Invariant 2, since accesses to $z_{j1,M}[M_1, \dots, M_{l'}]$ occur in M and satisfy Invariant 2.)

– When $x \in S$ occurs at the root of a term M_k in a condition *defined*$(M_1, \dots, M_l)$, we replace its definition *new* $x : T; Q$ with *let* $x : T = cst$ *in* Q for some constant cst; when it does not occur in *defined* tests, we remove its definition. If $x = z_{j1,M}$ for some M, we add *new* $z'_{jk,M} : T'_{jk,M}$ for each $k \leq \mathrm{nNew}'_{j,M}$ where *new* $x : T$ was.

– For each term $M \in \mathcal{M}$, let $P_M = C_M[M]$ be the smallest process containing M. (Note that M never occurs in an input, so P_M is an output process.) Let $l = \mathrm{nNewSeq}_M$. We replace P_M with $(\mathit{new}\ z'_{lk,M} : T'_{lk,M};)_{k \leq \mathrm{nNew}'_{l,M}} P'_M$ if $\mathrm{nNew}_{l,M} = 0$ and $\mathrm{nNew}'_{l,M} > 0$, with P'_M otherwise, where
– $P'_M = (\mathit{let}\ \mathrm{varImR}(x_{k,M}, M) : T_{k,M} = \sigma_M x_{k,M}\ \mathit{in}\ )_{k \leq \mathrm{nInput}_M} \mathrm{transf}_{\phi_0, C_M}(FP_M)$.
– ϕ_0 is defined as follows:

$$\phi_0(x_{j,M}[i_1, \dots, i_l]) = \mathrm{varImR}(x_{j,M}, M)[i'_1, \dots, i'_{l'}]$$
$$\phi_0(z[i_1, \dots, i_l]) = \mathrm{varImR}(z, M)[i'_1, \dots, i'_{l'}]$$
$$\phi_0(y'_{jk,M}[i_1, \dots, i_j]) = \mathrm{varImR}(y'_{jk,M}, M)[\mathrm{im\ index}_j(M)]$$

where $i_1, \dots, i_l$ are the current replication indexes at the definition of $x_{j,M}$ in R, $i'_1, \dots, i'_{l'}$ are the current replication indexes at M in Q_0, and z is a variable defined by *let* or *new* in FP_M.
– A function ϕ from array accesses to array accesses is extended to terms as substitution, by $\phi(f(M_1, \dots, M_m)) = f(\phi(M_1), \dots, \phi(M_m))$.
– $\mathrm{transf}_{\phi, C_M}(FP)$ is defined recursively as follows:

$$\mathrm{transf}_{\phi,C_M}(M') = C_M[\phi(M')]$$
$$\mathrm{transf}_{\phi,C_M}(\mathit{new}\ z : T; FP') = \mathit{new}\ \mathrm{varImR}(z, M) : T; \mathrm{transf}_{\phi,C_M}(FP')$$
$$\mathrm{transf}_{\phi,C_M}(\mathit{let}\ z : T = M'\ \mathit{in}\ FP') = \mathit{let}\ \mathrm{varImR}(z, M) : T = \phi(M')\ \mathit{in}\ \mathrm{transf}_{\phi,C_M}(FP')$$
$$\mathrm{transf}_{\phi,C_M}(\mathit{if}\ M_1\ \mathit{then}\ FP_1\ \mathit{else}\ FP_2) = \mathit{if}\ \phi(M_1)\ \mathit{then}\ \mathrm{transf}_{\phi,C_M}(FP_1)\ \mathit{else}\ \mathrm{transf}_{\phi,C_M}(FP_2)$$
$$\mathrm{transf}_{\phi,C_M}(\mathit{find}(\bigoplus_{j=1}^{m} FB_j)\ \mathit{else}\ FP') = \mathit{find}(\bigoplus_{j=1}^{m} \mathrm{transf}_{\phi,C_M}(FB_j))\ \mathit{else}\ \mathrm{transf}_{\phi,C_M}(FP')$$
$$\mathrm{transf}_{\phi,C_M}(\widetilde{u} \leq \widetilde{n}\ \mathit{suchthat}\ \mathit{defined}(z_k[M_{k1}, \dots, M_{kl'_k}]_{1 \leq k \leq l}) \wedge M_1\ \mathit{then}\ FP') =$$
$$\bigoplus_{M' \in \mathcal{M}'} \widetilde{u'} \leq \widetilde{n'}\ \mathit{suchthat}\ \mathit{defined}(\phi'(z_k[M_{k1}, \dots, M_{kl'_k}])_{1 \leq k \leq l}) \wedge$$
$$\mathrm{im\ index}_{j_1}(M')\{\widetilde{u'}/\widetilde{i'}\} = \mathrm{im\ index}_{j_1}(M) \wedge \phi'(M_1)\ \mathit{then}\ \mathrm{transf}_{\phi',C_M}(FP')$$

where j_1 is the length of the prefix of the current replication indexes that occurs in $M_{k1}, \dots, M_{kl'_k}$ (by hypothesis H7); $\mathcal{M}'$ is the set of $M' \in \mathcal{M}$ such that $\mathrm{varImR}(z_k, M')$ is defined for $k \leq l$ and M' and M share the first j_1 sequences of random variables; $\widetilde{i'}$ is the sequence of current replication indexes at M'; $\widetilde{u'}$ is a sequence formed with a fresh variable for each variable in $\widetilde{i'}$; $\widetilde{n'}$ is the sequence of bounds of replications above M'; ϕ' is an extension of ϕ with $\phi'(z_k[M_{k1}, \dots, M_{kl'_k}]) = \mathrm{varImR}(z_k, M')[\mathrm{im\ index}_j(M')\{\widetilde{u'}/\widetilde{i'}\}]$ if $z_k = y'_{jk',M'}$ for some k', and $\phi'(z_k[M_{k1}, \dots, M_{kl'_k}]) = \mathrm{varImR}(z_k, M')[\widetilde{u'}]$ if z_k is defined by *let* or by a function input. Optimizations for the definition of $\mathrm{transf}_{\phi,C_M}(FB)$ are presented in Appendix C.

The transformation essentially consists in two parts. First, add the restrictions to create random variables that correspond to random variables of R. We create the variables $z'_{jk,M}$ at the place where $z_{j1,M}$ was created in the initial game (We could have chosen $z_{jk',M}$ for any k'.), or when there is no $z_{j1,M}$, we have $j = \mathrm{nNewSeq}_M$ and we create $z'_{jk,M}$ just before evaluating M. Second, we transform the term M itself into the corresponding functional process of R, FP_M. The only delicate part for evaluating FP_M is the case of *find*: instead of looking up arrays of R, we look up the corresponding arrays of Q'_0 given by the mapping ϕ.

The following proposition shows the soundness of the transformation. It is proved in Appendix B.5.

Proposition 4. *Let Q_0 be a process that satisfies Invariants 1, 2, and 3, and Q'_0 the process obtained from Q_0 by the above transformation. Then Q'_0 satisfies Invariants 1, 2, and 3, and if $[\![L]\!] \approx [\![R]\!]$ for all polynomials* $\mathrm{maxlen}_\eta(c_{j_0,\dots,j_l})$ *and $I_\eta(n)$ where n is any replication bound of L or R, then $Q_0 \approx^V Q'_0$.*

We compute the possible values of the sets S and $\mathcal{M}$ by fixpoint iteration. We start with $\mathcal{M} = \emptyset$ and S containing a single variable of Q_0 bound by a restriction. (We try all possible variables.) When a term M of Q_0 contains a variable in S, we try to find a function in L that corresponds to M, and if we succeed, we add M to $\mathcal{M}$, and add to S variables in M that correspond to variables bound by restrictions in L. (If we fail, the transformation is not possible.) We continue until a fixpoint is reached, in which case all occurrences of variables of S are in terms of $\mathcal{M}$.

Example 4. In order to treat Example 1, the prover is given as input the indication that T_{mr}, T_r, T'_r, and T_k are fixed-length types; the type declarations for the functions $\mathit{mkgen}, \mathit{mkgen}' : T_{mr} \to T_{mk}$, $\mathit{mac}, \mathit{mac}' : \mathit{bitstring} \times T_{mk} \to T_{ms}$, $\mathit{check}, \mathit{check}' : \mathit{bitstring} \times T_{mk} \times T_{ms} \to \mathit{bool}$, $\mathit{kgen}, \mathit{kgen}' : T_r \to T_k$, $\mathit{enc}, \mathit{enc}' : \mathit{bitstring} \times T_k \times T'_r \to T_e$, $\mathit{dec} : T_e \times T_k \to \mathit{bitstring}_\perp$, $\mathit{k2b} : T_k \to \mathit{bitstring}$, $i_\perp : \mathit{bitstring} \to \mathit{bitstring}_\perp$, $Z : \mathit{bitstring} \to \mathit{bitstring}$, and the constant $Z_k : \mathit{bitstring}$; the equations (mac), (mac'), (enc), and $\forall x : T_k, Z(\mathit{k2b}(x)) = Z_k$ (which expresses that all keys have the same length); the indication that $\mathit{k2b}$ and $i_\perp$ are poly-injective (which generates the equations ($\mathit{k2b}$) and similar equations for $i_\perp$); equivalences $L \approx R$ for mac ($\mathit{mac}_{\mathrm{eq}}$) and encryption ($\mathit{enc}_{\mathrm{eq}}$); and the process Q_0 of Example 1.

The prover first applies $\mathbf{RemoveAssign}(x_{mk})$ to the process Q_0 of Example 1, as described in Example 2. The process can then be transformed using the security of the mac. We take $S = \{x'_r\}$, $M_1 = \mathit{mac}(x_m[i], \mathit{mkgen}(x'_r))$, $M_2 = \mathit{check}(x'_m[i'], \mathit{mkgen}(x'_r), x_{ma}[i'])$, and $\mathcal{M} = \{M_1, M_2\}$. We have $N_{M_1} = \mathit{mac}(x[i'', i], \mathit{mkgen}(r[i'']))$, $N_{M_2} = \mathit{check}(m[i'', i'], \mathit{mkgen}(r[i'']), \mathit{ma}[i'', i'])$, $\mathit{mapIdx}_{M_1}(a_1) = (1, a_1)$, and $\mathit{mapIdx}_{M_2}(a_2) = (1, a_2)$, so $x_m[a_1]$ corresponds to $x[1, a_1]$, x'_r to $r[1]$, $x'_m[a_2]$ to $m[1, a_2]$, and $x_{ma}[a_2]$ to $\mathit{ma}[1, a_2]$.

After transformation, we obtain the following process Q'_0:

$$\begin{aligned}
Q'_0 &= \mathit{start}(); \mathit{new}\ x_r : T_r; \mathit{let}\ x_k : T_k = \mathit{kgen}(x_r)\ \mathit{in}\ \mathit{new}\ x'_r : T_{mr}; \overline{c}\langle\rangle; (Q'_A \mid Q'_B)\\
Q'_A &= !^{i \le n} c_A[i](); \mathit{new}\ x'_k : T_k; \mathit{new}\ x''_r : T'_r; \mathit{let}\ x_m : \mathit{bitstring} = \mathit{enc}(\mathit{k2b}(x'_k), x_k, x''_r)\ \mathit{in}\\
&\qquad \overline{c_A[i]}\langle x_m, \mathit{mac}'(x_m, \mathit{mkgen}'(x'_r))\rangle\\
Q'_B &= !^{i' \le n} c_B[i'](x'_m, x_{ma}); \mathit{find}\ u \le n\ \mathit{suchthat}\ \mathit{defined}(x_m[u]) \wedge x'_m = x_m[u] \wedge\\
&\qquad \mathit{check}'(x'_m, \mathit{mkgen}'(x'_r), x_{ma})\ \mathit{then}\ (\mathit{if}\ 1\ \mathit{then}\ \mathit{let}\ i_\perp(\mathit{k2b}(x''_k)) = \mathit{dec}(x'_m, x_k)\ \mathit{in}\ \overline{c_B[i']}\langle\rangle)\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathit{else}\ (\mathit{if}\ 0\ \mathit{then}\ \mathit{let}\ i_\perp(\mathit{k2b}(x''_k)) = \mathit{dec}(x'_m, x_k)\ \mathit{in}\ \overline{c_B[i']}\langle\rangle)
\end{aligned}$$

The initial definition of x'_r is removed and replaced with a new definition, which we still call x'_r. The term $mac(x_m, mkgen(x'_r))$ is replaced with $mac'(x_m, mkgen'(x'_r))$. The term $check(x'_m, mkgen(x'_r), x_{ma})$ becomes $\mathit{find}\ u \leq n\ \mathit{suchthat}\ \mathit{defined}(x_m[u]) \wedge x'_m = x_m[u] \wedge check'(x'_m, mkgen'(x'_r), x_{ma})\ \mathit{then}\ 1\ \mathit{else}\ 0$ which yields Q'_B after transformation of functional processes into processes. The process looks up the message x'_m in the array x_m, which contains the messages whose mac has been computed with key $mkgen(x'_r)$. If the mac of x'_m has never been computed, the check always fails (it returns 0) by the definition of security of the mac. Otherwise, it returns 1 when $check'(x'_m, mkgen'(x'_r), x_{ma})$.

After applying **Simplify**, Q'_A is unchanged and Q'_B becomes

$$\begin{aligned} Q'_B = {}& !^{i' \leq n} c_B[i'](x'_m, x_{ma}); \mathit{find}\ u \leq n\ \mathit{suchthat}\ \mathit{defined}(x_m[u], x'_k[u]) \wedge x'_m = x_m[u] \wedge \\ & check'(x'_m, mkgen'(x'_r), x_{ma})\ \mathit{then}\ \mathit{let}\ x''_k : T_k = x'_k[u]\ \mathit{in}\ \overline{c_B[i']}\langle\rangle \end{aligned}$$

First, the tests *if* 1 *then* and *if* 0 *then* are simplified. The term $dec(x'_m, x_k)$ is simplified knowing $x'_m = x_m[u]$ by the *find* condition, $x_m[u] = enc(k2b(x'_k[u]), x_k, x''_r[u])$ by the assignment that defines x_m, $x_k = kgen(x_r)$ by the assignment that defines x_k, and $dec(enc(m, kgen(r), r'), kgen(r)) = i_\perp(m)$ by (enc). So we have $dec(x'_m, x_k) = i_\perp(k2b(x'_k[u]))$. By injectivity of $i_\perp$ and $k2b$, the assignment to x''_k simply becomes $x''_k = x'_k[u]$, using the equations $\forall x : \mathit{bitstring}, i_\perp^{-1}(i_\perp(x)) = x.$ and $\forall x : T_k, k2b^{-1}(k2b(x)) = x$.

After applying **RemoveAssign**(x_k), one can apply the security of encryption: $enc(k2b(x'_k), kgen(x_r), x''_r)$ becomes $enc'(Z(k2b(x'_k)), kgen(x_r), x''_r)$. After **Simplify**, it becomes $enc'(Z_k, kgen(x_r), x''_r)$, using $\forall x : T_k, Z(k2b(x)) = Z_k$ (which expresses that all keys have the same length).

Using lists instead of arrays simplifies this transformation: we do not need to add instructions that insert values in the list, since all variables are always implicitly arrays. Moreover, if there are several occurrences of $mac(x_i, k)$ with the same key in the initial process, each $check(m_j, k, ma_j)$ is replaced with a *find* with one branch for each occurrence of *mac*. Therefore, the prover distinguishes automatically the cases in which the checked mac ma_j comes from each occurrence of *mac*, that is, it distinguishes cases depending on the value of i such that $m_j = x_i$. Typically, distinguishing these cases is useful in the following of the proof. (A similar situation arises for other cryptographic primitives specified using *find*.)

4 Criteria for Proving Secrecy Properties

Let us now define syntactic criteria that allow us to prove secrecy properties of protocols. We use $\mathit{if}\ \mathit{defined}(M)\ \mathit{then}\ P$ as syntactic sugar for $\mathit{find}\ \ \mathit{suchthat}\ \mathit{defined}(M) \wedge 1\ \mathit{then}\ P\ \mathit{else}\ \overline{\mathit{yield}}\langle\rangle$.

Definition 5 (One-session secrecy). *The process* Q preserves the one-session secrecy of x *when* $Q \mid Q_x \approx Q \mid Q'_x$, *where*

$$\begin{aligned} Q_x &= c(u_1 : [1, n_1], \dots, u_m : [1, n_m]); \mathit{if}\ \ \mathit{defined}(x[u_1, \dots, u_m])\ \mathit{then}\ \overline{c}\langle x[u_1, \dots, u_m]\rangle \\ Q'_x &= c(u_1 : [1, n_1], \dots, u_m : [1, n_m]); \mathit{if}\ \ \mathit{defined}(x[u_1, \dots, u_m])\ \mathit{then}\ \mathit{new}\ y : T; \overline{c}\langle y\rangle \end{aligned}$$

$c \notin \mathrm{fc}(Q)$, $u_1, \dots, u_m \notin \mathrm{var}(Q)$, *and* $\mathcal{E}(x) = [1, n_1] \times \ldots \times [1, n_m] \to T$.

Intuitively, the adversary cannot distinguish a process that outputs the value of the secret from one that outputs a random number. The adversary performs a single test query, modeled by Q_x and Q'_x.

Proposition 5 (One-session secrecy). *Consider a process Q such that there exists a set of variables S such that 1) the definitions of x are either restrictions new $x[\widetilde{i}] : T$ and $x \in S$, or assignments let $x[\widetilde{i}] : T = z[M_1, \dots, M_l]$ where z is defined by restrictions new $z[i'_1, \dots, i'_l] : T$, and $z \in S$, and 2) all accesses to variables $y \in S$ in Q are of the form "let $y'[\widetilde{i}] : T' = y[M_1, \dots, M_l]$" with $y' \in S$. Then $Q \mid Q_x \approx_0 Q \mid Q'_x$, hence Q preserves the one-session secrecy of x.*

Intuitively, only the variables in S depend on the restriction that defines x; the sent messages and the control flow of the process are independent of x, so the adversary obtains no information on x. In the implementation, the set S is computed by fixpoint iteration, starting from x or z and adding variables y' defined by "*let* $y'[\widetilde{i}] : T' = y[M_1, \dots, M_l]$" when $y \in S$.

Definition 6 (Secrecy). *The process* Q preserves the secrecy of x *when* $Q \mid R_x \approx Q \mid R'_x$*, where*

$$R_x = !^{i \leq n} c(u_1 : [1, n_1], \dots, u_m : [1, n_m]); \textit{if defined}(x[u_1, \dots, u_m]) \textit{ then } \overline{c}\langle x[u_1, \dots, u_m]\rangle$$
$$R'_x = !^{i \leq n} c(u_1 : [1, n_1], \dots, u_m : [1, n_m]); \textit{if defined}(x[u_1, \dots, u_m]) \textit{ then find } u' \leq n \textit{ suchthat}$$
$$\textit{defined}(y[u'], u_1[u'], \dots, u_m[u']) \wedge u_1[u'] = u_1 \wedge \dots \wedge u_m[u'] = u_m \textit{ then } \overline{c}\langle y[u']\rangle \textit{ else new } y : T; \overline{c}\langle y\rangle$$

$c \notin \mathrm{fc}(Q)$*,* $u_1, \dots, u_m, u' \notin \mathrm{var}(Q)$*,* $\mathcal{E}(x) = [1, n_1] \times \dots \times [1, n_m] \to T$*, and* $I_\eta(n) \geq I_\eta(n_1) \times \dots \times I_\eta(n_m)$*.*

Intuitively, the adversary cannot distinguish a process that outputs the value of the secret for several indexes from one that outputs independent random numbers. In this definition, the adversary can perform several test queries, modeled by R_x and R'_x. This corresponds to the "real-or-random" definition of security [4]. (As shown in [4], this notion is stronger than the more standard approach in which the adversary can perform a single test query and some reveal queries, which always reveal $x[u_1, \dots, u_m]$.)

Proposition 6 (Secrecy). *Assume that* Q *satisfies the hypothesis of Proposition 5.*

When $\mathcal{T}$ *is a trace of* $C[Q]$ *for some evaluation context* C*, we define* $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}])$*, the defining restriction of* $x[\widetilde{a}]$ *in trace* $\mathcal{T}$*, as follows: if* $x[\widetilde{a}]$ *is defined by new* $x[\widetilde{a}] : T$ *in* $\mathcal{T}$*,* $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) = x[\widetilde{a}]$*; if* $x[\widetilde{a}]$ *is defined by let* $x[\widetilde{a}] : T = z[M_1, \dots, M_l]$*,* $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) = z[a'_1, \dots, a'_l]$ *where* $E, M_k \Downarrow a'_k$ *for all* $k \leq l$ *and* E *is the environment in* $\mathcal{T}$ *at the definition of* $x[\widetilde{a}]$*.*

Assume that for all evaluation contexts C *acceptable for* Q*,* 0*,* $\{x\}$*, the probability* $\Pr[\mathcal{T} \wedge \widetilde{a} \neq \widetilde{a'} \wedge \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) = \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a'}])]$ *is negligible. Then* Q *preserves the secrecy of* x*.*

Intuitively, the required condition guarantees that if $\widetilde{a} \neq \widetilde{a'}$, then $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) \neq \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a'}])$ except in cases of negligible probability, so $x[\widetilde{a}]$ and $x[\widetilde{a'}]$ are defined by different restrictions so they are independent random numbers.

In order to check this condition, we use the following algorithm. For each definition P of x, we define $\mathrm{defRestr}_P(x[i_1, \dots, i_m])$ as follows:

$$\mathrm{defRestr}_P(x[i_1, \dots, i_m]) = \begin{cases} x[i_1, \dots, i_m] & \text{if } P = \textit{new } x[i'_1, \dots, i'_m] : T; P' \\ z[M_1, \dots, M_l]\{i_1/i'_1, \dots, i_m/i'_m\} & \\ & \text{if } P = \textit{let } x[i'_1, \dots, i'_m] : T = z[M_1, \dots, M_l] \textit{ in } P' \end{cases}$$

We also define $\mathrm{TrueFacts}_P[i_1, \dots, i_m] = \mathrm{TrueFacts}_P\{i_1/i'_1, \dots, i_m/i'_m\}$ in both cases. For each pair of definitions of x, P, P', let $\mathrm{defRestr}_P(x[i_1, \dots, i_m]) = z[M_1, \dots, M_l]$ and $\mathrm{defRestr}_{P'}(x[i'_1, \dots, i'_m]) = z'[M'_1, \dots, M'_{l'}]$. We check that, if $z = z'$ then $\mathrm{TrueFacts}_P[i_1, \dots, i_m] \cup \mathrm{TrueFacts}_{P'}[i'_1, \dots, i'_m] \cup \{i_1 \neq i'_1 \vee \dots \vee i_m \neq i'_m, M_1 = M'_1, \dots, M_l = M'_l\}$ yields a contradiction by the equational prover described in Section 3.2. When this check succeeds, the second condition holds. Indeed, this means that $\mathrm{defRestr}_P(x[i_1, \dots, i_m]) = \mathrm{defRestr}_{P'}(x[i'_1, \dots, i'_m])$ and $(i_1, \dots, i_m) \neq (i'_1, \dots, i'_m)$ yield a contradiction (taking into account the facts $\mathrm{TrueFacts}_P[i_1, \dots, i_m]$ $\mathrm{TrueFacts}_{P'}[i'_1, \dots, i'_m]$ that are known to hold when x is defined by the considered definitions). So when $(i_1, \dots, i_m) \neq (i'_1, \dots, i'_m)$, $x[i_1, \dots, i_m]$ and $x[i'_1, \dots, i'_m]$ are defined by different restrictions, as desired.

This notion of secrecy composed with correspondence assertions [45] can be used to prove security of a key exchange. (Correspondence assertions are properties of the form "if some event $e(\widetilde{M})$ has been executed then some events $e_i(\widetilde{M_i})$ for $i \leq m$ have been executed".) We postpone this point to a future paper, since we do not present the verification of correspondence assertions in this paper. (This verification is currently being implemented.)

Lemma 2. *If $Q \approx^{\{x\}} Q'$ and Q preserves the one-session secrecy of x then Q' preserves the one-session secrecy of x. The same result holds for secrecy.*

We can then apply the following technique. When we want to prove that Q_0 preserves the (one-session) secrecy of x, we transform Q_0 by the transformations described in Section 3 with $V = \{x\}$. By Propositions 1, 2, and 4, we obtain a process Q'_0 such that $Q_0 \approx^V Q'_0$. We use Propositions 5 or 6 to show that Q'_0 preserves the (one-session) secrecy of x, and finally conclude that Q_0 also preserves the (one-session) secrecy of x by Lemma 2.

Example 5. After the transformations of Example 4, the only variable access to x'_k in the considered process is $\mathit{let}\ x''_k : T_k = x'_k[u]$ and x''_k is not used in the considered process. So by Proposition 5, the considered process preserves the one-session secrecy of x''_k (with $S = \{x'_k, x''_k\}$). By Lemma 2, the process of Example 1 also preserves the one-session secrecy of x''_k. However, this process does not preserve the secrecy of x''_k, because the adversary can force several sessions of B to use the same key x''_k, by replaying the message sent by A. (Accordingly, the hypothesis of Proposition 6 is not satisfied.)

The criteria given in this section might seem restrictive, but in fact, they should be sufficient for all protocols, provided the previous transformation steps are powerful enough to transform the protocol into a simpler protocol, on which these criteria can then be applied.

5 Proof Strategy

Up to now, we have described the available game transformations. Next, we explain how we organize these transformations in order to prove protocols.

The prover uses a simplification procedure for games that applies the transformations **Simplify**, **MoveNew**, **RemoveAssign(useless)**, and **SArename(auto)**. At the beginning of the proof, and after each successful cryptographic transformation (that is, a transformation of Section 3.3), the prover executes this simplification procedure, and tests whether the desired security properties are proved, as described in Section 4. If so, it stops.

In order to perform the cryptographic transformations and the other syntactic transformations, our proof strategy relies of the idea of advice. Precisely, the prover tries to execute each available cryptographic transformation in turn. When such a cryptographic transformation fails, it returns some syntactic transformations that could make the desired transformation work. (These are the advised transformations.) Then the prover tries to perform these syntactic transformations. If they fail, they may also suggest other advised transformations, which are then executed. When the syntactic transformations finally succeed, we retry the desired cryptographic transformation, which may succeed or fail, perhaps with new advised transformations, and so on.

The prover determines the advised transformations as follows:

- Assume that we try to execute a cryptographic transformation, and need to recognize a certain term M of L, but we find in Q_0 only part of M, the other parts being variable accesses $x[\dots]$ while we expect function applications. In this case, we advise **RemoveAssign**(x). For example, if Q_0 contains $\mathit{enc}(M', x_k, x'_r)$ and we look for $\mathit{enc}(x_m, \mathit{kgen}(x_r), x_{r'})$, we advise **RemoveAssign**(x_k). If Q_0 contains $\mathit{let}\ x_k = \mathit{mkgen}(x_r)$ and we look for $\mathit{mac}(x_m, \mathit{mkgen}(x_r))$, we advise **RemoveAssign**(x_k). (The transformation of Example 2 is advised for this reason.)
- When we try to execute **RemoveAssign**(x), x has several definitions, and there are accesses to variable x guarded by *find* in Q_0, we advise **SArename**(x).
- When we check whether x is secret or one-session secret, we have an assignment $\mathit{let}\ x[\widetilde{i}] : T = y[\widetilde{M}]\ \mathit{in}\ P$, and there is at least one assignment defining y, we advise **RemoveAssign**(y).

When we check whether x is secret or one-session secret, we have an assignment *let* $x[\widetilde{i}] : T = y[\widetilde{M}]$ *in* P, y is defined by restrictions, y has several definitions, and some variable accesses to y are not of the form *let* $y'[\widetilde{i'}] : T = y[\widetilde{M'}]$ *in* P', then we advise **SArename**(y).

These pieces of advice were sufficient in the examples we tried, but one may obviously extend them if needed.

6 Experimental Results

We have successfully tested our prover on a number of protocols of literature. All these protocols have been tested in a configuration in which the honest participants are willing to run sessions with the adversary, and we prove secrecy of keys for sessions between honest participants. In these examples, shared-key encryption is encoded using a stream cipher and a mac as in Example 1, public-key encryption is assumed to be IND-CCA2 (indistinguishability under adaptive chosen-ciphertext attacks) [13], public-key signature is assumed to be secure against existential forgery.

- Otway-Rees [39]: We automatically prove the secrecy of the exchanged key.
- Yahalom [17]: For the original version of the protocol, our prover cannot show one-session secrecy of the exchanged key, because the protocol is not secure, at least using encrypt-then-mac as definition of encryption. Indeed, there is a confirmation round $\{N_B\}_K$ where K is the exchanged key. This message may reveal some information on K. After removing this confirmation round, our prover shows the one-session secrecy of K. However, it cannot show the secrecy of K, since in the absence of a confirmation round, the adversary may force several sessions of Yahalom to use the same key.
- Needham-Schroeder shared-key [37]: Our prover shows one-session secrecy of the exchanged key. It does not prove the secrecy of the exchanged key, since there is a well known attack [22] in which the adversary forces several sessions of the protocol to use the same key. Our prover shows the secrecy for the corrected version [38].
- Denning-Sacco public-key [22]: Our prover cannot show the one-session secrecy of the exchanged key, since there is an attack against this protocol [2]. One-session secrecy of the exchanged key is proved for the corrected version [2]. Secrecy is not proved since the adversary can force several sessions of the protocol to use the same key. (We do not model timestamps in this protocol. In contrast to the previous examples, we give the main proof steps to the prover manually, as follows:

  ```
  SArename Rkey
  crypto enc rkB
  crypto sign rkS
  crypto sign rkA
  success
  ```

  The variable `Rkey` defines a table of public keys, and is assigned at three places, corresponding to principals A and B, and to other principals defined by the adversary. The instruction `SArename Rkey` allows us to distinguish these three cases. The instruction `crypto enc rkB` means that the prover should apply the definition of security of encryption (primitive `enc`), for the key generated from random number `rkB`. The instruction `success` means that prover should check whether the desired security properties are proved.
- Needham-Schroeder public-key [37]: This protocol is an authentication protocol. Since our prover cannot check authentication yet, we transform it into a key exchange protocol in several ways, by choosing for the key either one of the nonces N_A and N_B shared between A and B, or $H(N_A, N_B)$ where H is a hash function (in the random oracle model). When the key is $H(N_A, N_B)$, one-session secrecy of the key cannot be proved for the original protocol, due to the well-known attack [32].

For the corrected version [32], our prover shows secrecy of the key. For both versions, the prover cannot prove one-session secrecy of N_A or N_B. For N_B, the failure of the proof corresponds to an attack: the adversary can check whether it is given N_B or a random number by sending $\{N'_B\}_{pk_B}$ to B as the last message of the protocol: B accepts if and only if $N'_B = N_B$. For N_A, the failure of the proof comes from limitations of our prover: The prover cannot take into account that N_A is accepted only after all messages that contain N_A have been sent, which prevents the previous attack. (This is the only case in our examples where the failure of the proof comes from limitations of the prover. This problem could probably be solved by improving the transformation **Simplify**.) Like for the Denning-Sacco protocol, we provided the main proof steps to the prover manually, as follows when the distributed key is N_A or N_B:

```
SArename Rkey
crypto sign rkS
crypto enc rkA
crypto enc rkB
success
```

When the distributed key is $H(N_A, N_B)$, the proof is as follows:

```
SArename Rkey
crypto sign rkS
crypto enc rkA
crypto enc rkB
crypto hash
SArename Na_39
simplify
success
```

The total runtime for all these tests is 60 s on a Pentium M 1.8 GHz.

7 Conclusion

This paper presents a prover for cryptographic protocols sound in the computational model. This prover works with no or very little help from the user, can handle a wide variety of cryptographic primitives in a generic way, and produces proofs valid for a polynomial number of sessions in the presence of an active adversary. Thus, it represents important progress with respect to previous work in this area.

We have recently extended our prover to provide exact security proofs (that is, proofs with an explicit probability of an attack, instead of the asymptotic result that this probability is negligible) and to prove correspondence assertions. We leave these extensions for a future paper. In the future, it would also be interesting to handle even more cryptographic primitives, such as Diffie-Hellman key agreements. (In order to handle them, the language of equivalences that we use to specify the security properties of primitives will need to be extended.)

Acknowledgments I warmly thank David Pointcheval for his advice and explanations of the computational proofs of protocols. This project would not have been possible without him. This work was partly supported by ARA SSIA Formacrypt.

References

1. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In N. Kobayashi and B. Pierce, editors, *Theoretical Aspects of Computer Software (TACS'01)*, volume 2215 of *Lecture Notes on Computer Science*, pages 82–94, Sendai, Japan, Oct. 2001. Springer.

2. M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, Jan. 1996.
3. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
4. M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In S. Vaudenay, editor, *Proceedings of the 2005 International Workshop on Practice and Theory in Public Key Cryptography (PKC'05)*, volume 3386 of *Lecture Notes on Computer Science*, pages 65–84, Les Diablerets, Switzerland, Jan. 2005. Springer.
5. P. Adão, G. Bana, J. Herzog, and A. Scedrov. Soundness of formal encryption in the presence of key-cycles. In S. de Capitani di Vimercati, P. Syverson, and D. Gollmann, editors, *Proceedings of the 10th European Symposium On Research In Computer Security (ESORICS 2005)*, volume 3679 of *Lecture Notes on Computer Science*, pages 374–396, Milan, Italy, Sept. 2005. Springer.
6. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *17th IEEE Computer Security Foundations Workshop*, Pacific Grove, CA, June 2004. IEEE.
7. M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. In *26th IEEE Symposium on Security and Privacy*, pages 171–182, Oakland, CA, May 2005. IEEE.
8. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *10th ACM conference on Computer and communication security (CCS'03)*, pages 220–230, Washington D.C., Oct. 2003. ACM.
9. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In E. Snekkenes and D. Gollman, editors, *Computer Security - ESORICS 2003, 8th European Symposium on Research in Computer Security*, volume 2808 of *Lecture Notes on Computer Science*, pages 271–290, Gjøovik, Norway, Oct. 2003. Springer.
10. G. Barthe, J. Cederquist, and S. Tarento. A machine-checked formalization of the generic model and the random oracle model. In D. Basin and M. Rusinowitch, editors, *Second International Joint Conference on Automated Reasoning (IJCAR'04)*, volume 3097 of *Lecture Notes on Computer Science*, pages 385–399, Cork, Ireland, July 2004. Springer.
11. M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In L. Caires and L. Monteiro, editors, *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes on Computer Science*, pages 652–663, Lisboa, Portugal, July 2005. Springer.
12. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of the 38th Symposium on Foundations of Computer Science (FOCS'97)*, pages 394–403, Miami Beach, Florida, Oct. 1997. IEEE. Full paper available at `http://www-cse.ucsd.edu/users/mihir/papers/sym-enc.html`.
13. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In H. Krawczyk, editor, *Advances in Cryptology - CRYPTO '98*, volume 1462 of *Lecture Notes on Computer Science*, pages 26–45, Santa Barbara, California, USA, Aug. 1998. Springer.
14. M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, Dec. 2000.
15. M. Bellare and P. Rogaway. The game-playing technique. Cryptology ePrint Archive, Report 2004/331, Dec. 2004. Available at `http://eprint.iacr.org/2004/331`.
16. B. Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy*, pages 86–100, Oakland, California, May 2004.
17. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, February 1989.
18. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, Las Vegas, Nevada, Oct. 2001. IEEE. An updated version is available at Cryptology ePrint Archive, `http://eprint.iacr.org/2000/067`.
19. R. Canetti and J. Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). Cryptology ePrint Archive, Report 2004/334, 2004. Available at `http://eprint.iacr.org/2004/334`.

20. V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In M. Sagiv, editor, *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes on Computer Science*, pages 157–171, Edimbourg, U.K., Apr. 2005. Springer.
21. A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In L. Caires and L. Monteiro, editors, *ICALP 2005: the 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes on Computer Science*, pages 16–29, Lisboa, Portugal, July 2005. Springer.
22. D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24(8):533–536, Aug. 1981.
23. S. Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, June 2005. Available at `http://eprint.iacr.org/2005/181`.
24. J. Herzog. A computational interpretation of Dolev-Yao adversaries. In R. Gorrieri, editor, *WITS'03 - Workshop on Issues in the Theory of Security*, pages 146–155, Warsaw, Poland, Apr. 2003.
25. R. Janvier, Y. Lakhnech, and L. Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In M. Sagiv, editor, *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes on Computer Science*, pages 172–185, Edimbourg, U.K., Apr. 2005. Springer.
26. D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, U.K., 1970.
27. 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 *Lecture Notes on Computer Science*, pages 159–173, Warsaw, Poland, Apr. 2003. Springer.
28. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *IEEE Symposium on Security and Privacy*, pages 71–85, Oakland, California, May 2004.
29. P. Laud. Secrecy types for a simulatable cryptographic library. In *12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 26–35, Alexandria, VA, Nov. 2005. ACM.
30. P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *ACM Computer and Communication Security (CCS-5)*, pages 112–121, San Francisco, California, Nov. 1998.
31. P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security protocols. In J. Wing, J. Woodcock, and J. Davies, editors, *FM'99 World Congress On Formal Methods in the Development of Computing Systems*, volume 1708 of *Lecture Notes on Computer Science*, pages 776–793, Toulouse, France, Sept. 1999. Springer.
32. 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 *Lecture Notes on Computer Science*, pages 147–166. Springer, 1996.
33. P. Mateus, J. Mitchell, and A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus. In R. Amadio and D. Lugiez, editors, *CONCUR 2003 - Concurrency Theory, 14-th International Conference*, volume 2761 of *Lecture Notes on Computer Science*, pages 327–349, Marseille, France, Sept. 2003. Springer.
34. D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004.
35. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In M. Naor, editor, *Theory of Cryptography Conference (TCC'04)*, volume 2951 of *Lecture Notes on Computer Science*, pages 133–151, Cambridge, MA, USA, Feb. 2004. Springer.
36. J. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for the analysis of cryptographic protocols. Submitted. Available at `http://theory.stanford.edu/people/jcm/publications.htm`, 2004.
37. R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, Dec. 1978.
38. R. M. Needham and M. D. Schroeder. Authentication revisited. *Operating Systems Review*, 21(1):7, 1987.
39. D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.
40. A. Ramanathan, J. Mitchell, A. Scedrov, and V. Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In I. Walukiewicz, editor, *FOSSACS 2004 - Foundations of Software Science and Computation Structures*, volume 2987 of *Lecture Notes on Computer Science*, pages 468–483, Barcelona, Spain, Mar. 2004. Springer.

41. V. Shoup. A proposal for an ISO standard for public-key encryption, Dec. 2001. ISO/IEC JTC 1/SC27.
42. V. Shoup. OAEP reconsidered. *Journal of Cryptology*, 15(4):223–249, Sept. 2002.
43. C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner. Cryptographically sound theorem proving. Unpublished manuscript, Feb. 2006.
44. S. Tarento. Machine-checked security proofs of cryptographic signature schemes. In S. de Capitani di Vimercati, P. Syverson, and D. Gollmann, editors, *Proceedings of the 10th European Symposium On Research In Computer Security (ESORICS 2005)*, volume 3679 of *Lecture Notes on Computer Science*, pages 140–158, Milan, Italy, Sept. 2005. Springer.
45. 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, Oakland, California, May 1993.

Appendix

A Modeling other Primitives

This appendix gives the definition of a number of cryptographic primitives in our prover.

A.1 Extensions

Guiding the Application of Equivalences We introduce a small extension to the equivalences $(G_1, \dots, G_m) \approx (G'_1, \dots, G'_m)$ described in Section 3.3. These equivalences become $(G_1\ \mathit{mode}_1, \dots, G_m\ \mathit{mode}_m) \approx (G'_1, \dots, G'_m)$, where mode_j is either empty or $[\mathit{all}]$. The mode $[\mathit{all}]$ is an indication for the prover, to guide the application of the equivalence without changing its semantics. When $\mathit{mode}_j = [\mathit{all}]$, $\mathcal{M}$ must contain all occurrences in the initial game Q of the root function symbols of terms M inside G_j. When mode_j is empty, at least one variable defined by *new* in G_j must correspond to a variable in S.

The following hypotheses guarantee the good usage of modes:

H8. At most one mode_j can be empty. (Otherwise, when several sets of random variables can be chosen for each G_j, there are many possible combinations for applying the transformation.)
H9. If G_j is of the form $!^{i \leq n}(x_1 : T_1, \dots, x_l : T_l) \rightarrow FP$ without any restriction, then $\mathit{mode}_j = [\mathit{all}]$. (A restriction is needed in the definition of empty mode.)

Relaxing Hypothesis H6 Hypothesis H6 requires that for all restrictions *new* $y : T$ that occur above a term N in the left-hand side of an equivalence, y occurs in N. We can relax this hypothesis, by allowing that some random variables y do not occur in N, provided that the missing variables can be determined using Hypothesis H′4.1: when some term M shares some variable y in the l'-th sequence of random variables with some other term M', we know that it must also share with M' all random variables in sequences above and including the l'-th sequence; so, knowing the random variables associated to M', we can determine some of those associated to M. The transformation simply fails when the algorithm described above cannot fully determine the random variables associated to some term M.

Relaxing Hypothesis H′2 Hypothesis H′2 requires that no term N transformed by the equivalence occurs in the condition part of a *find* $(\mathit{defined}(M_1, \dots, M_l) \wedge M)$. We can relax this hypothesis by allowing N to occur in M (but not in the *defined* test), provided the variables $\widetilde{u}$ bound by this *find* do not occur in the following terms in the transformed expression of N:

- N' in processes of the form *let* $x : T = N'$ *in* … or *if* N' *then* … *else* …;
- N'_{jk} and N'_j in processes of the form $\mathit{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathit{suchthat}\ \mathit{defined}(N'_{j1}, \dots, N'_{jl_j}) \wedge N'_j\ \mathit{then}\ \dots)\ \mathit{else}\ \dots$.

(If the variables $\widetilde{u}$ bound by *find* occurred in such terms, the transformation would move them outside the scope of their definition.)

A.2 Public-key cryptography

Signature

$$\begin{array}{l}
T_r \text{ large, fixed length}; T'_r \text{ fixed length}\\
s, s' : T \times T_{sk} \times T'_r \rightarrow T_s\\
c, c' : T \times T_{pk} \times T_s \rightarrow \mathit{bool}\\
\mathit{skgen}, \mathit{skgen}' : T_r \rightarrow T_{sk}\\
\mathit{pkgen}, \mathit{pkgen}' : T_r \rightarrow T_{pk}
\end{array}$$

$$\begin{array}{l}
\forall m : T, \forall r : T_r, \forall r' : T'_r, c(m, \mathit{pkgen}(r), s(m, \mathit{skgen}(r), r')) = 1\\
\forall m : T, \forall r : T_r, \forall r' : T'_r, c'(m, \mathit{pkgen}'(r), s'(m, \mathit{skgen}'(r), r')) = 1\\
\mathit{new}\ x : T_r; \mathit{new}\ y : T_r; f(x) = f(y) \approx x = y \text{ for } f \in \{\mathit{pkgen}, \mathit{skgen}, \mathit{pkgen}', \mathit{skgen}'\}
\end{array}$$

```
$!^{i'''\leq n'''} \mathit{new}\ r : T_r;($
    $!^{i\leq n}() \rightarrow \mathit{pkgen}(r),$
    $!^{i'\leq n'} \mathit{new}\ r' : T'_r; (x : T) \rightarrow s(x, \mathit{skgen}(r), r')),$
$!^{i''\leq n''}(m : T, y : T_{pk}, si : T_s) \rightarrow c(m, y, si)\ [\mathit{all}]$
  $\approx$
1. $!^{i'''\leq n'''} \mathit{new}\ r : T_r;($
2.     $!^{i\leq n}() \rightarrow \mathit{pkgen}'(r),$
3.     $!^{i'\leq n'} \mathit{new}\ r' : T'_r; (x : T) \rightarrow s'(x, \mathit{skgen}'(r), r')),$
4. $!^{i''\leq n''}(m : T, y : T_{pk}, si : T_s) \rightarrow$
5.     $\mathit{find}\ u \leq n''', u' \leq n'\ \mathit{suchthat}\ \mathit{defined}(r[u], x[u, u'])\wedge$
6.                 $y = \mathit{pkgen}'(r[u]) \wedge m = x[u, u'] \wedge c'(m, y, si)\ \mathit{then}\ 1\ \mathit{else}$
7.     $\mathit{find}\ u \leq n'''\ \mathit{suchthat}\ \mathit{defined}(r[u]) \wedge y = \mathit{pkgen}'(r[u])\ \mathit{then}\ 0\ \mathit{else}$
8.     $c(m, y, si)$
```

The first three lines of each side of the equivalence express that the generation of public keys and the computation of the signature are left unchanged in the transformation. The verification of a signature $c(m, y, si)$ is replaced with a lookup in the previously computed signatures: if the signature is checked using one of the keys $\mathit{pkgen}'(r[u])$ (that is, if $y = \mathit{pkgen}'(r[u])$), then it can be valid only when it has been computed by the signature oracle $s'(x, \mathit{skgen}'(r[u]), r')$, that is, when $m = x[u, u']$ for some u'. Lines 5-6 of the right-hand side of the equivalence try to find such a u' and return 1 when they succeed. Line 7 of the right-hand side returns 0 when no such u' is found in lines 5-6, but $y = \mathit{pkgen}'(r[u])$ for some u. The last line handles the case when the key y is not $\mathit{pkgen}'(r[u])$. In this case, we check the signature as before. (Using c and not c' in the last line of the transformation allows to reapply this transformation with another value of r.)

We can model deterministic signatures in a similar way, by removing the third argument of s.

IND-CCA2 Public-Key Encryption

$$\begin{array}{l}
T_r \text{ large, fixed length}; T'_r \text{ fixed length}\\
\mathit{enc}, \mathit{enc}' : T \times T_{pk} \times T'_r \rightarrow T_e\\
\mathit{dec}, \mathit{dec}' : T_e \times T_{sk} \rightarrow T_\perp
\end{array}$$

$skgen, skgen' : T_r \to T_{sk}$
$pkgen, pkgen' : T_r \to T_{pk}$
$i_\perp : T \to T_\perp$ (poly-injective)

$\forall m : T, \forall r : T_r, \forall r' : T'_r, dec(enc(m, pkgen(r), r'), skgen(r)) = i_\perp(m)$
$\forall m : T, \forall r : T_r, \forall r' : T'_r, dec'(enc'(m, pkgen'(r), r'), skgen'(r)) = i_\perp(m)$
$new\ x : T_r; new\ y : T_r; f(x) = f(y) \approx x = y$ for $f \in \{pkgen, pkgen', skgen, skgen'\}$

```
$!^{i'''\leq n'''} new\ r : T_r; ($
   $!^{i\leq n}() \to pkgen(r),$
   $!^{i'\leq n'}(m : T_e) \to dec(m, skgen(r))),$
$!^{i''\leq n''} new\ r' : T'_r; (x : T, y : T_{pk}) \to enc(x, y, r')\ [all]$
 $\approx$
$!^{i'''\leq n'''} new\ r : T_r; ($
   $!^{i\leq n}() \to pkgen'(r),$
   $!^{i'\leq n'}(m : T_e) \to find\ u \leq n''\ suchthat\ defined(m'[u], x[u], y[u]) \wedge y[u] = pkgen'(r) \wedge m = m'[u]\ then$
      $i_\perp(x[u])\ else\ dec'(m, skgen'(r))),$
$!^{i''\leq n''}(x : T, y : T_{pk}) \to$
      $find\ u' \leq n'''\ suchthat\ defined(r[u']) \wedge y = pkgen'(r[u'])\ then$
         $let\ m' : T_e = new\ r' : T'_r; enc'(Z_T, pkgen'(r[u']), r')\ in\ m'$
      $else\ new\ r'' : T'_r; enc(x, y, r'')$
```

When no decryption is present, this transformation reduces to IND-CPA public key encryption below.

IND-CPA Public-Key Encryption

T_r large, fixed length; T'_r fixed length
$enc, enc' : T \times T_{pk} \times T'_r \to T_e$
$dec : T_e \times T_{sk} \to T_\perp$
$skgen : T_r \to T_{sk}$
$pkgen, pkgen' : T_r \to T_{pk}$
$i_\perp : T \to T_\perp$ (poly-injective)

$\forall m : T, \forall r : T_r, \forall r' : T'_r, dec(enc(m, pkgen(r), r'), skgen(r)) = i_\perp(m)$
$new\ x : T_r; new\ y : T_r; f(x) = f(y) \approx x = y$ for $f \in \{pkgen, skgen, skgen'\}$

```
$!^{i\leq n} new\ r : T_r; () \to pkgen(r),$
$!^{i'\leq n'} new\ r' : T'_r; (x : T, y : T_{pk}) \to enc(x, y, r')\ [all]$
 $\approx$
$!^{i\leq n} new\ r : T_r; () \to pkgen'(r),$
$!^{i'\leq n'}(x : T, y : T_{pk}) \to find\ u \leq n\ suchthat\ defined(r[u]) \wedge y = pkgen'(r[u])\ then$
      $new\ r' : T'_r; enc'(Z_T, pkgen'(r[u]), r')\ else\ new\ r'' : T'_r; enc(x, y, r'')$
```

A.3 Hash functions

Collision Resistant Hash Function

$$\begin{array}{l} T_k \text{ fixed length} \\ h : T_k \times \mathit{bitstring} \to T \\ \mathit{new}\ k : T_k; \forall x : \mathit{bitstring}, y : \mathit{bitstring}, h(k,x) = h(k,y) \approx x = y \end{array}$$

Hash Function in the Random Oracle Model

$$\begin{array}{l} T \text{ fixed length} \\ h : \mathit{bitstring} \to T \\ !^{i \leq n}(x : \mathit{bitstring}) \to h(x)\ [\mathit{all}] \\ \approx_0 \\ !^{i \leq n}(x : \mathit{bitstring}) \to \mathit{find}\ u \leq n\ \mathit{suchthat}\ \mathit{defined}(x[u], r[u]) \wedge x = x[u]\ \mathit{then}\ r[u]\ \mathit{else}\ \mathit{new}\ r : T; r \end{array}$$

Note that the game must include, in parallel with the protocol to verify, the process $!^{i \leq n} c(x : \mathit{bitstring}); \overline{c}\langle h(x) \rangle$. Otherwise, the prover would incorrectly assume that the adversary cannot compute the hash function. This particularity is related to the fact that a random oracle is unimplementable: otherwise, the adversary could implement it without being explicitly given access to it.

A.4 Xor

$$\begin{array}{l} \mathit{xor} : T \times T \to T \text{ (commutative)} \\ \forall x : T, y : T, \mathit{xor}(x, \mathit{xor}(x,y)) = y. \\ !^{i \leq n} \mathit{new}\ k : T; (x : T) \to \mathit{xor}(x,k) \\ \approx_0 \\ !^{i \leq n} \mathit{new}\ k : T; (x : T) \to k \end{array}$$

B Proofs

B.1 Proofs for Section 2.3

Property P1: Each Variable is Defined at Most Once We define the multiset of variable accesses that may be defined by a process as follows:

$$\begin{array}{l} \mathit{Defined}(0) = \emptyset \\ \mathit{Defined}(Q_1 \mid Q_2) = \mathit{Defined}(Q_1) \uplus \mathit{Defined}(Q_2) \\ \mathit{Defined}(!^{i \leq n} Q) = \uplus_{a \in [1, I_\eta(n)]} \mathit{Defined}(Q\{a/i\}) \\ \mathit{Defined}(\mathit{newChannel}\ c; Q) = \mathit{Defined}(Q) \\ \mathit{Defined}(c[M_1, \ldots, M_l](x_1[\widetilde{a}] : T_1, \ldots, x_k[\widetilde{a}] : T_k); P) = \{x_j[\widetilde{a}] \mid j \leq k\} \uplus \mathit{Defined}(P) \\ \mathit{Defined}(\overline{c[M_1, \ldots, M_l]}\langle N_1, \ldots, N_k \rangle; Q) = \mathit{Defined}(Q) \\ \mathit{Defined}(\mathit{new}\ x[\widetilde{a}] : T; P) = \{x[\widetilde{a}]\} \uplus \mathit{Defined}(P) \\ \mathit{Defined}(\mathit{let}\ x[\widetilde{a}] : T = M\ \mathit{in}\ P) = \{x[\widetilde{a}]\} \uplus \mathit{Defined}(P) \\ \mathit{Defined}(\mathit{if}\ M\ \mathit{then}\ P\ \mathit{else}\ P') = \max(\mathit{Defined}(P), \mathit{Defined}(P')) \end{array}$$

$$\mathit{Defined}(\mathit{find}\ (\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{a}] \leq \widetilde{n_j}\ \mathit{suchthat}\ \mathit{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathit{then}\ P_j)\ \mathit{else}\ P) =$$
$$\max(\max_{j=1}^{m} \mathit{Defined}(P_j), \mathit{Defined}(P))$$

We define $\mathit{Defined}(E) = \mathrm{Dom}(E)$, $\mathit{Defined}(E, P, \mathcal{Q}, \mathcal{C}) = \mathit{Defined}(E) \uplus \mathit{Defined}(P) \uplus \uplus_{Q \in \mathcal{Q}} \mathit{Defined}(Q)$.

Invariant 4 (Single definition, for executing games). The semantic configuration $E, P, \mathcal{Q}, \mathcal{C}$ satisfies Invariant 4 if and only if $\mathit{Defined}(E, P, \mathcal{Q}, \mathcal{C})$ does not contain duplicate elements.

Lemma 3. *If Q_0 satisfies Invariant 1, then* $\mathrm{initConfig}(Q_0)$ *satisfies Invariant 4.*

Lemma 4. *If $E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E', P', \mathcal{Q}', \mathcal{C}'$ with $p > 0$ and $E, P, \mathcal{Q}, \mathcal{C}$ satisfies Invariant 4, then so does $E', P', \mathcal{Q}', \mathcal{C}'$.*

Proof sketch. We show by cases following the definition of $\xrightarrow{p}_t$ that if $E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E', P', \mathcal{Q}', \mathcal{C}'$ then $\mathit{Defined}(E, P, \mathcal{Q}, \mathcal{C}) \subseteq \mathit{Defined}(E', P', \mathcal{Q}', \mathcal{C}')$. The result follows. □

Property P1 follows easily. Indeed, by Invariant 4, just before executing a definition of $x[\widetilde{a}]$, $\mathit{Defined}(E, P, \mathcal{Q}, \mathcal{C})$ does not contain duplicate elements, so $x[\widetilde{a}] \notin \mathrm{Dom}(E)$ since $x[\widetilde{a}] \in \mathit{Defined}(P) \uplus \mathit{Defined}(\mathcal{Q})$. So each variable is defined at most once for each value of its array indexes in a trace of Q_0.

Property P2: Variables are Defined Before Being Used

Invariant 5 (Defined variables, for executing games). The semantic configuration $E, P, \mathcal{Q}, \mathcal{C}$ satisfies Invariant 5 if and only if each occurrence of a variable access $x[M_1, \dots, M_m]$ in P or $\mathcal{Q}$ is either

- present in $\mathrm{Dom}(E)$: for all $j \leq m$, $E, M_j \Downarrow a_j$ and $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$;
- or syntactically under the definition of $x[M_1, \dots, M_m]$ (in which case for all $j \leq m$, M_j is a constant or variable replication index);
- or in a *defined* condition in a *find* process;
- or in M'_j or P_j in a process of the form *find* $(\bigoplus_{j=1}^{m''} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}$ *suchthat defined*$(M'_{j1}, \dots, M'_{jl_j}) \wedge M'_j$ *then* $P_j)$ *else* P where for some $k \leq l_j$, $x[M_1, \dots, M_m]$ is a subterm of M'_{jk}.

Lemma 5. *If Q_0 satisfies Invariant 2, then* $\mathrm{initConfig}(Q_0)$ *satisfies Invariant 5.*

Lemma 6. *If $E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E', P', \mathcal{Q}', \mathcal{C}'$ with $p > 0$ and $E, P, \mathcal{Q}, \mathcal{C}$ satisfies Invariant 5, then so does $E', P', \mathcal{Q}', \mathcal{C}'$.*

Proof sketch. If $x[M_1, \dots, M_m]$ is in the second case, and we execute the definition of $x[M_1, \dots, M_m]$, then for all $j \leq m$, M_j is a constant replication index and $x[M_1, \dots, M_m]$ is added to $\mathrm{Dom}(E)$ by rules (New), (Let), (Find1), or (Output), so it moves to the first case.

If $x[M_1, \dots, M_m]$ is in the third case, and we execute the corresponding *find*, this access to x simply disappears.

If $x[M_1, \dots, M_m]$ is in the last case, and we execute the *find* selecting branch j, then $x[M_1, \dots, M_m]$ is a subterm of M'_{jk} for $k \leq l_j$. We easily show by induction on M that, if $E, M \Downarrow a$, then for all subterms $x[M_1, \dots, M_m]$ of M, for all $j' \leq m$, $E, M'_{j'} \Downarrow a_{j'}$ and $x[a_1, \dots, a_m]$ is in $\mathrm{Dom}(E)$. Therefore, by hypothesis of the semantic rule for *find*, for all $j' \leq m$, $E, M_{j'} \Downarrow a_{j'}$ and $x[a_1, \dots, a_m]$ is in $\mathrm{Dom}(E)$. So $x[M_1, \dots, M_m]$ also moves to the first case.

In all other cases, the situation remains unchanged. □

Property P2 follows easily. Indeed, consider an application of rule (Var), where the array access $x[M_1, \dots, M_m]$ is not in a *defined* condition of a *find*. Then, this array access is not under any variable definition or *find*, so for all $j \leq m$, $E, M_j \Downarrow a_j$ and $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$. Hence, the test $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$ succeeds.

Property P3: Typing We say that $\mathcal{E} \vdash_\eta E$ if and only if $E(x[a_1, \ldots, a_m]) = a$ implies $\mathcal{E}(x) = T_1 \times \ldots \times T_m \to T$ with for all $j \leq m$, $a_j \in I_\eta(T_j)$ and $a \in I_\eta(T)$. We define $\mathcal{E} \vdash_\eta P$ as $\mathcal{E} \vdash P$, $\mathcal{E} \vdash_\eta Q$ as $\mathcal{E} \vdash Q$, and $\mathcal{E} \vdash_\eta M : T$ as $\mathcal{E} \vdash M : T$, with the additional rule $\mathcal{E} \vdash_\eta a : T$ if and only if $a \in I_\eta(T)$. (This rule is useful to type constant replication indexes. In the formulas giving the typing rules, replication indexes i may then also be constants a.) We say that $\mathcal{E} \vdash_\eta E, P, \mathcal{Q}, \mathcal{C}$ if and only if $\mathcal{E} \vdash_\eta E$, $\mathcal{E} \vdash_\eta P$, and for all $Q \in \mathcal{Q}$, $\mathcal{E} \vdash_\eta Q$. Similarly, $\mathcal{E} \vdash_\eta E, \mathcal{Q}, \mathcal{C}$ if and only if $\mathcal{E} \vdash_\eta E$ and for all $Q \in \mathcal{Q}$, $\mathcal{E} \vdash_\eta Q$.

Lemma 7. *If $\mathcal{E} \vdash_\eta E$, $\mathcal{E} \vdash_\eta M : T$, and $E, M \Downarrow a$, then $\mathcal{E} \vdash_\eta a : T$*

Proof sketch. By induction on the derivation of $E, M \Downarrow a$. □

Lemma 8. *If $\mathcal{E} \vdash_\eta E, \mathcal{Q}, \mathcal{C}$ and $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E', \mathcal{Q}', \mathcal{C}'$, then $\mathcal{E} \vdash_\eta E', \mathcal{Q}', \mathcal{C}'$.*
So, if $\mathcal{E} \vdash_\eta E, \mathcal{Q}, \mathcal{C}$, then $\mathcal{E} \vdash_\eta \mathrm{reduce}(E, \mathcal{Q}, \mathcal{C})$.

Proof sketch. By cases on the derivation of $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E', \mathcal{Q}', \mathcal{C}'$. In the case of the replication, we use a substitution lemma, noticing that $a \in I_\eta([1, n])$, so $\mathcal{E} \vdash_\eta a : [1, n]$. In the case of the input, we use Lemma 7. □

Lemma 9. *If $\mathcal{E} \vdash Q_0$, then $\mathcal{E} \vdash_\eta \mathrm{initConfig}(Q_0)$.*

Proof sketch. By Lemma 8 and the previous definitions. □

Lemma 10 (Subject reduction). *If $\mathcal{E} \vdash_\eta E, P, \mathcal{Q}, \mathcal{C}$ and $E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E', P', \mathcal{Q}', \mathcal{C}'$ with $p > 0$, then $\mathcal{E} \vdash_\eta E', P', \mathcal{Q}', \mathcal{C}'$.*

Proof sketch. By cases on the derivation of $E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E', P', \mathcal{Q}', \mathcal{C}'$, using Lemmas 7 and 8. □

Property P3 is an immediate consequence of Lemmas 10 and 7.

Property P4: Runtime We give a very brief sketch of this proof here. We refer the reader to [36] for a more detailed proof for a different calculus; their proof could be adapted to our calculus.

The length of all bitstrings manipulated by processes is polynomial in the security parameter η. Indeed, by hypothesis, the length of received messages is limited by maxlen_η, so polynomial in the security parameter η. The length of random bitstrings is also polynomial in the security parameter by hypothesis on the types. Function symbols correspond to functions that run in polynomial time, so they output bitstrings of size polynomial in the size of their inputs, so also polynomial in the security parameter.

Since the number of copies generated by each replication is polynomial in the security parameter, the total number of executed instructions is polynomial in the security parameter, and it is easy to see that each instruction runs in polynomial time since bitstrings are of polynomial length. Therefore, processes run in polynomial time.

B.2 Proof of Proposition 1

Proof sketch of Proposition 1 The proof that Q_0' satisfies Invariants 1, 2, and 3 is in general easy, and the proof of $Q_0 \approx_0^V Q_0'$ relies on a correspondence between traces of $C[Q_0]$ and traces of $C[Q_0']$, with the same probability and such that a configuration of the trace of $C[Q_0]$ executes $\overline{c}\langle a \rangle$ immediately if and only if the corresponding configuration of the corresponding trace of $C[Q_0']$ executes $\overline{c}\langle a \rangle$ immediately. This correspondence is obtained by replacing some internal actions of Q_0 with corresponding internal actions of Q_0'. We sketch the proof only for the case of **SArename**(x) and leave the other cases to the reader.

The process Q'_0 satisfies Invariant 1 because definitions of variables duplicated by **SArename** all occur in a different branch of a *find*.

For Invariant 2, each variable access $x_j[M_1, \dots, M_m]$ in Q'_0 comes from a variable access $x[M_1, \dots, M_m]$ in Q_0. Since Q_0 satisfies Invariant 2, either this access is under its definition, in which case **SArename**(x) has replaced this definition of x with a definition of x_j, so $x_j[M_1, \dots, M_m]$ is under its definition in Q'_0; or this access is in a *defined* test, in which case it is also in a *defined* test in Q'_0; or this access is in a branch of *find* with a condition *defined*$(N_1, \dots, N_l)$ such that $x[M_1, \dots, M_m]$ is a subterm of N_j for some $j \leq l$, in which case $x[M_1, \dots, M_m]$ has been substituted with $x_j[M_1, \dots, M_m]$ in this branch of *find*, so $x_j[M_1, \dots, M_m]$ is under a suitable *defined* condition. Therefore Q'_0 satisfies Invariant 2.

For Invariant 3, the type environment $\mathcal{E}'$ for Q'_0 is obtained from the type environment $\mathcal{E}$ for Q_0, by setting $\mathcal{E}'(x_1) = \dots = \mathcal{E}'(x_m) = \mathcal{E}(x)$ and $\mathcal{E}'(x)$ is not defined. (Indeed, all definitions of x in Q_0 have the same type $\mathcal{E}(x)$, which is therefore the type of the definitions of $x_j, j \leq m$ in Q'_0.) The proof of $\mathcal{E}' \vdash Q'_0$ is obtained from the proof of $\mathcal{E} \vdash Q_0$, by replacing requests to $\mathcal{E}(x)$ with requests to $\mathcal{E}(x_j)$ for some $j \leq m$, and duplicating parts of the proof of $\mathcal{E} \vdash Q_0$ that correspond to duplicated branches of *find*.

Finally, let us prove that $Q_0 \approx^V_0 Q'_0$. We denote by $\mathit{SArename}(x, Q)$ the process obtained by applying **SArename**(x) to Q. Let j be a partial function from l-tuples of indexes $a_1, \dots, a_l$ to subscripts of variable x. Informally, j is such that $x[a_1, \dots, a_l]$ in a trace of Q_0 corresponds to $x_{j(a_1,\dots,a_l)}[a_1, \dots, a_l]$ in the corresponding trace of Q'_0. We define a function $\mathit{SArename}_j$ that relates configurations in a trace of Q_0 to configurations in a trace of the renamed process Q'_0.

- We define $\mathit{SArename}_j$ for processes as follows: $\mathit{SArename}_j(x, E, Q_1)$ first computes $\mathit{SArename}(x, Q_1) = Q_2$. More precisely, it renames each definition of x to the name used when renaming the whole process Q_0; it replaces variable accesses to x with variable accesses to x_j when the definition of x that caused this replacement in Q_0 also occurs in Q_1; it duplicates branches of *find* as $\mathit{SArename}(x, Q_0)$, renaming variable accesses to x into variable accesses to x_j when the *find* that caused this replacement in Q_0 also occurs in Q_1. (When a variable access to x is under both a definition of x and *find*, or under several nested *finds* that guarantee that it is defined, it is important to follow exactly the renaming procedure that happened in Q_0. Formally, this can be done by annotating each construct in processes with a distinct occurrence symbol and by reducing annotated processes. When we perform $\mathit{SArename}(x, Q_0)$, we can then remember the occurrence symbols of the constructs that cause each variable renaming.) Finally, $\mathit{SArename}_j$ replaces all remaining occurrences of $x[M_1, \dots, M_l]$ in Q_2 such that $E, M_k \Downarrow a_k$ for $k \leq l$ and $x[a_1, \dots, a_l] \in \mathrm{Dom}(E)$, with $x_{j(a_1,\dots,a_l)}[M_1, \dots, M_l]$. $\mathit{SArename}_j$ is defined similarly for terms ($\mathit{SArename}$ has no effect in this case) and for processes.
- We also define $\mathit{SArename}_j$ for environments: $E' = \mathit{SArename}_j(x, E)$ if and only if $E'(x_{j(a_1,\dots,a_l)}[a_1, \dots, a_l]) = E(x[a_1, \dots, a_l])$ when $x[a_1, \dots, a_l] \in \mathrm{Dom}(E)$, $E'(y[a_1, \dots, a_l]) = E(y[a_1, \dots, a_l])$ when $y \neq x$ and $y[a_1, \dots, a_l] \in \mathrm{Dom}(E)$, E' is undefined in all other cases.
- We extend $\mathit{SArename}_j$ to semantic configurations: $\mathit{SArename}_j(x, (E, P, \mathcal{Q}, \mathcal{C})) = (\mathit{SArename}_j(x, E), \mathit{SArename}_j(x, E, P), \{\mathit{SArename}_j(x, E, Q_1) \mid Q_1 \in \mathcal{Q}\}, \mathcal{C})$. We also define $\mathit{SArename}_j(x, (E, \mathcal{Q}, \mathcal{C}))$ in the same way.

We first show that if $E, M \Downarrow a$, then $\mathit{SArename}_j(x, E), \mathit{SArename}_j(x, E, M) \Downarrow a$. The proof proceeds by induction on M. The only interesting case is $M = x[M_1, \dots, M_l]$. Since $E, M \Downarrow a$ has been derived by (Var), $E, M_k \Downarrow a_k$ for all $k \leq l$ and $a = E(x[a_1, \dots, a_l])$. By induction hypothesis, $\mathit{SArename}_j(x, E), \mathit{SArename}_j(x, E, M_k) \Downarrow a_k$ for all $k \leq l$. Moreover, $\mathit{SArename}_j(x, E, x[M_1, \dots, M_l]) = x_{j(a_1,\dots,a_l)}[\mathit{SArename}_j(x, E, M_1), \dots, \mathit{SArename}_j(x, E, M_l)]$ and $\mathit{SArename}_j(x, E)(x_{j(a_1,\dots,a_l)}[a_1, \dots, a_l]) = E(x[a_1, \dots, a_l]) = a$, so $\mathit{SArename}_j(x, E), \mathit{SArename}_j(x, E, M) \Downarrow a$.

Next, we can easily show by cases on the reduction $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E', \mathcal{Q}', \mathcal{C}'$ that, if $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E', \mathcal{Q}', \mathcal{C}'$, then $\mathit{SArename}_j(x, (E, \mathcal{Q}, \mathcal{C})) \rightsquigarrow \mathit{SArename}_j(x, (E', \mathcal{Q}', \mathcal{C}'))$. Hence $\mathit{SArename}_j(x, \mathrm{reduce}(E, \mathcal{Q}, \mathcal{C})) = \mathrm{reduce}(\mathit{SArename}_j(x, (E, \mathcal{Q}, \mathcal{C})))$.

Let C be any evaluation context acceptable for Q_0, Q'_0, V. We show that for each trace $\mathrm{initConfig}(C[Q_0]) \rightarrow_\eta \ldots \rightarrow_\eta E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$, there exists a trace $\mathrm{initConfig}(C[Q'_0]) \rightarrow_\eta \ldots \rightarrow_\eta E'_m, P'_m, \mathcal{Q}'_m, \mathcal{C}_m$ with the same probability, and a function j_m such that $E'_m, P'_m \mathcal{Q}'_m, \mathcal{C}_m = \mathit{SArename}_{j_m}(x, (E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m))$. The proof proceeds by induction on the length m of the trace. For the induction step, we distinguish cases depending on the last reduction step of the trace.

- Initial case $m = 0$: $\mathrm{fc}(C[Q_0]) = \mathrm{fc}(C[Q'_0])$ since the transformation **SArename** does not modify channels. Let j_0 be the function defined nowhere. We have, $C[Q'_0] = \mathit{SArename}_{j_0}(x, \emptyset, C[Q_0])$. Indeed, since $x \notin V$, $x \notin \mathrm{var}(C)$, so $\mathit{SArename}_{j_0}(x, \emptyset, C[Q_0]) = \mathit{SArename}(x, C[Q_0]) = C[\mathit{SArename}(x, Q_0)] = C[Q'_0]$. Therefore, $\mathit{SArename}_{j_0}(x, (\emptyset, \{C[Q_0]\}, \mathrm{fc}(C[Q_0]))) = (\emptyset, \{C[Q'_0]\}, \mathrm{fc}(C[Q'_0]))$. Hence we have $\mathit{SArename}_{j_0}(x, \mathrm{reduce}(\emptyset, \{C[Q_0]\}, \mathrm{fc}(C[Q_0]))) = \mathrm{reduce}(\emptyset, \{C[Q'_0]\}, \mathrm{fc}(C[Q'_0]))$. Thus, $\mathit{SArename}_{j_0}(x, \mathrm{initConfig}(C[Q_0])) = \mathrm{initConfig}(C[Q'_0])$.
- The last step of the trace is a definition of $x[a_1, \ldots, a_l]$: By induction hypothesis, we have a trace of length $m - 1$, with an associated function j_{m-1}. Since $C[Q_0]$ satisfies Invariant 1, the configuration $E_{m-1}, P_{m-1}, \mathcal{Q}_{m-1}, \mathcal{C}_{m-1}$ satisfies Invariant 4, so $x[a_1, \ldots, a_l] \notin \mathrm{Dom}(E_{m-1})$. Since $P'_{m-1} = \mathit{SArename}_{j_{m-1}}(x, E_{m-1}, P_{m-1})$, the first instruction of P'_{m-1} is a definition of $x_k[a_1, \ldots, a_l]$ for some k (using the property shown above to prove that the indexes of x, resp. x_k, are the same in the execution of P_{m-1} and of P'_{m-1}). We define $j_m = j_{m-1}[(a_1, \ldots, a_l) \mapsto k]$, and easily show that we obtain a suitable trace of length m with this function j_m.
- The last step of the trace is a *find* whose *defined* condition refers to x: By induction hypothesis, we have a trace of length $m - 1$, with an associated function j_{m-1}. If a branch FB of the *find* in P_{m-1} succeeds for certain values of $\widetilde{i}$, exactly one of its copies succeeds in P'_{m-1}, the copy whose *defined* condition refers to $x_{j_{m-1}(a_1,\ldots,a_l)}[a_1, \ldots, a_l]$ when the *defined* condition of the branch FB in P_{m-1} refers to $x[a_1, \ldots, a_l]$. If a branch of the *find* fails in P_{m-1}, all its copies fail in P'_{m-1}. Therefore, the number $|S|$ of successful choices of the *find* is the same in P_{m-1} and in P'_{m-1}. Hence, the probability that each successful branch is taken is the same. When P_{m-1} executes a successful branch, we build the corresponding trace of P'_{m-1} by executing the successful copy of this branch. When P_{m-1} executes the *else* branch, P'_{m-1} also executes the *else* branch. So it is easy to see that we obtain a suitable trace of length m with associated function $j_m = j_{m-1}$ (except when the *find* also defines $x[a'_1, \ldots, a'_l]$, in which case the previous item of the proof must also be applied).
- All other cases are easy: they execute in the same way in P_{m-1} and in P'_{m-1}.

We also show the converse property, that for each trace $\mathrm{initConfig}(C[Q'_0]) \rightarrow_\eta \ldots \rightarrow_\eta E'_m, P'_m, \mathcal{Q}'_m, \mathcal{C}_m$, there exists a trace $\mathrm{initConfig}(C[Q_0]) \rightarrow_\eta \ldots \rightarrow_\eta E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ with the same probability and $E'_m, P'_m \mathcal{Q}'_m, \mathcal{C}_m = \mathit{SArename}_{j_m}(x, (E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m))$. The proof is similar to the proof above.

If $E'_m, P'_m \mathcal{Q}'_m, \mathcal{C}_m = \mathit{SArename}_{j_m}(x, (E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m))$, then for all channels c and bitstrings a, $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ executes $\overline{c}\langle a\rangle$ immediately if and only if $E'_m, P'_m, \mathcal{Q}'_m, \mathcal{C}_m$ executes $\overline{c}\langle a\rangle$ immediately. So $\Pr[C[Q_0] \rightsquigarrow_\eta \overline{c}\langle a\rangle] = \Pr[C[Q'_0] \rightsquigarrow_\eta \overline{c}\langle a\rangle]$. Therefore $Q_0 \approx_0^V Q'_0$. □

B.3 Proof of Proposition 2

Proof sketch of Proposition 2 The proof of Invariants 1, 2, and 3 is relatively easy, so we focus on the proof of $Q_0 \approx^V Q'_0$.

Let C be any evaluation context acceptable for Q_0, Q'_0, V. Let $q(\eta)$ be the maximum runtime of $C[Q_0]$, where q is a polynomial.

We define $p_{\max}(\eta) = \max(\{\frac{1}{|I_\eta(T)|} \mid T \text{ is a large type}\} \cup \{p(\eta) \text{ associated to user-defined rewrite rules, for an adversary of runtime } q(\eta)\})$. $p_{\max}(\eta)$ is negligible, since it is the maximum of a constant number of negligible functions. We shall prove in the following that the probability that a desired fact does not hold is at most $q'(\eta)p_{\max}(\eta)$, where q' is a polynomial, so it is negligible.

We consider a slightly modified semantics for our calculus, in which each process is accompanied with a substitution that defines the values of the replication indexes in that process. For example, the rule (Repl) becomes in this semantics:

$$E, \{(\sigma, !^{i \leq n} Q)\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow E, \{(\sigma[i \mapsto a], Q) \mid a \in [1, I_\eta(n)]\} \uplus \mathcal{Q}, \mathcal{C}$$

When evaluating a term M in a process with substitution (σ, Q) or (σ, P), we now use $E, \sigma, M \Downarrow a$ instead of $E, M \Downarrow a$, with the rule $E, \sigma, i \Downarrow \sigma i$ instead of (Cst), and the other rules modified accordingly.

The judgment $E, \sigma \vdash F$ means that a fact F holds in environment E and substitution σ. It is defined by $E, \sigma \vdash M$ if and only if $E, \sigma, M \Downarrow 1$ and $E, \sigma \vdash \mathit{defined}(M)$ if and only if $E, \sigma, M \Downarrow a$ for some a. We extend this definition to sets of facts naturally.

Let us consider the sets of facts $\mathrm{TrueFacts}_Q$, $\mathrm{TrueFacts}_P$ after calling collectFacts but before adding consequences of *defined* facts. For occurrences of processes P, Q in C, we let $\mathrm{TrueFacts}_P = \mathrm{TrueFacts}_Q = \emptyset$.

We first prove that if $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E', \mathcal{Q}', \mathcal{C}'$ and for all $(\sigma, Q) \in \mathcal{Q}$, $E, \sigma \vdash \mathrm{TrueFacts}_Q$, then for all $(\sigma, Q) \in \mathcal{Q}'$, $E', \sigma \vdash \mathrm{TrueFacts}_Q$. The proof is easy by cases on the derivation of $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E', \mathcal{Q}', \mathcal{C}'$. Therefore, if $E', \mathcal{Q}', \mathcal{C}' = \mathrm{reduce}(E, \mathcal{Q}, \mathcal{C})$ and for all $(\sigma, Q) \in \mathcal{Q}$, $E, \sigma \vdash \mathrm{TrueFacts}_Q$, then for all $(\sigma, Q) \in \mathcal{Q}'$, $E', \sigma \vdash \mathrm{TrueFacts}_Q$.

Next, we show that if $\mathrm{initConfig}(C[Q_0]) \xrightarrow{p}_t \ldots \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ then for all $(\sigma', P') \in \{(\sigma, P)\} \cup \mathcal{Q}$, $E, \sigma' \vdash \mathrm{TrueFacts}_{P'}$. The proof proceeds by induction on the length of the trace. For the initial configuration, the property follows immediately from the previous property of reduce. For the inductive step, if the reduced process is in C, the result is obvious since $\mathrm{TrueFacts}_{P'} = \emptyset$. Otherwise, we proceed by cases on the last reduction of the trace. For example, in the case (If1), $E, \sigma, M \Downarrow 1$ and $E, (\sigma, \mathit{if}\ M\ \mathit{then}\ P_1\ \mathit{else}\ P_2), \mathcal{Q}, \mathcal{C} \xrightarrow{1}_{I1} E, (\sigma, P_1), \mathcal{Q}, \mathcal{C}$. We have $\mathrm{TrueFacts}_{P_1} = \{M\} \cup \mathrm{TrueFacts}_{\mathit{if}\ M\ \mathit{then}\ P_1\ \mathit{else}\ P_2}$. By induction hypothesis, $E, \sigma \vdash \mathrm{TrueFacts}_{\mathit{if}\ M\ \mathit{then}\ P_1\ \mathit{else}\ P_2}$. Since $E, \sigma, M \Downarrow 1$, $E, \sigma \vdash M$, so $E, \sigma \vdash \mathrm{TrueFacts}_{P_1}$. We have $E, \sigma' \vdash \mathrm{TrueFacts}_{P'}$ for $(\sigma', P') \in \mathcal{Q}$ immediately by induction hypothesis. We proceed in a similar way for the other cases (using the property proved above for reduce in the (Output) case).

Let us now consider the sets of facts $\mathrm{TrueFacts}_Q$, $\mathrm{TrueFacts}_P$ after adding consequences of *defined* facts. We show that we still have if $\mathrm{initConfig}(C[Q_0]) \xrightarrow{p}_t \ldots \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ then for all $(\sigma', P') \in \{(\sigma, P)\} \cup \mathcal{Q}$, $E, \sigma' \vdash \mathrm{TrueFacts}_{P'}$. The proof follows the addition of consequences of *defined* facts. Assume that $\mathit{defined}(M) \in \mathrm{TrueFacts}_{P'}$, $x[M_1, \ldots, M_m]$ is a subterm of M, and we update $\mathrm{TrueFacts}_{P'}$ into

$$\mathrm{TrueFacts}'_{P'} = \mathrm{TrueFacts}_{P'} \cup (\cap_{(x[i_1,\ldots,i_m],\mathcal{F}) \in \mathrm{TrueFacts}_{\mathrm{def}}} \mathcal{F}\{M_1/i_1, \ldots, M_m/i_m\})$$

We assume that $E, \sigma' \vdash \mathrm{TrueFacts}_{P'}$ and show that $E, \sigma' \vdash \mathrm{TrueFacts}'_{P'}$. Since $E, \sigma' \vdash \mathit{defined}(M)$, $E, \sigma', M_j \Downarrow a_j$ for all $j \leq m$ and $x[a_1, \ldots, a_m] \in \mathrm{Dom}(E)$. Therefore, some definition of $x[a_1, \ldots, a_m]$ has been executed in the considered trace. We show that, for some $(x[i_1, \ldots, i_m], \mathcal{F}) \in \mathrm{TrueFacts}_{\mathrm{def}}$, we have $E, \sigma_1 \vdash \mathcal{F}$ where $\sigma_1(i_1) = a_1, \ldots, \sigma_1(i_m) = a_m$. The desired result then follows easily. Let $E_1, (\sigma_1, P_1), \mathcal{Q}_1, \mathcal{C}_1 \xrightarrow{p}_t E_2, (\sigma_1, P_2), \mathcal{Q}_2, \mathcal{C}_2$ be the reduction that defines $x[a_1, \ldots, a_m]$ in the considered trace. By the previous property, $E_2, \sigma_1 \vdash \mathrm{TrueFacts}_{P_2}$. (We consider here the value of $\mathrm{TrueFacts}_{P_2}$ just after collectFacts.) We have $(x[i_1, \ldots, i_m], \mathcal{F}) \in \mathrm{TrueFacts}_{\mathrm{def}}$ where $\mathcal{F} \subseteq \mathrm{TrueFacts}_{P_2}$, so $E, \sigma_1 \vdash \mathcal{F}$ since E is an extension of E_2 so all facts that hold in E_2 also hold in E.

Let us now show the correctness of the equational prover. We say that $E, \sigma \vdash (\mathcal{F}, \mathcal{R})$ when $E, \sigma \vdash \mathcal{F}$ and for all $(M_1 \to M_2) \in \mathcal{R}$, $E, \sigma \vdash M_1 = M_2$. We show that $\Pr[\mathrm{initConfig}(C[Q_0]) \to \ldots \to E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge (\mathcal{F}_0, \mathcal{R}_0) = (\mathrm{TrueFacts}_P, \emptyset) \wedge \forall j \leq m', \frac{\mathcal{F}_{j-1}, \mathcal{R}_{j-1}}{\mathcal{F}_j, \mathcal{R}_j} \wedge E, \sigma \not\vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})] \leq q'(\eta)p_{\max}(\eta)$ for some polynomial q'. The proof proceeds by induction on m'. For $m' = 0$, this is an immediate consequence of the previous result, with $q'(\eta) = 0$. For the inductive step, we proceed by cases on the derivation of $\frac{\mathcal{F}_{m'-1},\mathcal{R}_{m'-1}}{\mathcal{F}_{m'},\mathcal{R}_{m'}}$.

- The cases (2), (5), (7), as well as the cases (1) and (6) when the reduction uses a rule of $\mathcal{R}$, are obvious and there is no loss of probability (that is, $q'(\eta)$ is unchanged).
- Cases (1) and (6) when the reduction uses a user-defined rewrite rule *new* $y_1 : T'_1, \ldots,$ *new* $y_l : T'_l, \forall x_1 : T_1, \ldots, \forall x_m : T_m, M_1 \rightarrow M_2$, with associated probability $p(\eta)$: Assuming this user-defined claim is correct, the desired result holds with a loss of probability $p(\eta)$ times the number of possible values for the indexes of restrictions that correspond to $y_1, \ldots, y_l$, which is polynomial in η. Indeed, when $E, \sigma \vdash (\mathcal{F}_{m'-1}, \mathcal{R}_{m'-1})$ but $E, \sigma \not\vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})$, for at least one value of the indexes of restrictions that correspond to $y_1, \ldots, y_l$, the process $C[Q_0]$ provides an adversary that satisfies the conditions of the definition of the corresponding user claim. (The proof of Proposition 3 below details a similar argument in a more complicated case.)
- Case (3): Assume that $E, \sigma \vdash (\mathcal{F}_{m'-1}, \mathcal{R}_{m'-1})$ but $E, \sigma \not\vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})$ So for all $j \leq m$, $E, \sigma, M_j \Downarrow a_j$, $E, \sigma, M'_j \Downarrow a'_j$, $(a_1, \ldots, a_m) \neq (a'_1, \ldots, a'_m)$, and $E(x[a_1, \ldots, a_m]) = E(x[a_1, \ldots, a_m])$. Since for each $a_1, \ldots, a_m$, $x[a_1, \ldots, a_m]$ is chosen randomly among $|I_\eta(T)|$ values, the probability that this happens is smaller than $\frac{q''(\eta)(q''(\eta)-1)}{2|I_\eta(T)|}$ where $q''(\eta)$ is the number of possible values of $a_1, \ldots, a_m$, which is a polynomial in η.
- Case (4): We first show that, if M characterizes y, then there exists $\widetilde{M}$ such that for each a, there exists b such that for all E and σ, $E, \sigma, M \Downarrow a$ implies $E, \sigma, y[\widetilde{M}] \Downarrow b$. Indeed, $\mathcal{M}_0 = \{\alpha M = M\}$ is rewritten into a set that contains $(\alpha y)[\widetilde{M'}] = y[\widetilde{M}]$. Due to the form of rewrite rules, $(\alpha y)[\widetilde{M'}]$ is a subterm of αM and $y[\widetilde{M}]$ is a subterm of M.
  - If a is such that there exist E' and σ' such that $E', \sigma', \alpha M \Downarrow a$ and E', σ' define variables of αM, let b such that $E', \sigma', (\alpha y)[\widetilde{M'}] \Downarrow b$. Then for all E, σ such that $E, \sigma, M \Downarrow a$, we can define the E'', σ'' that map variables of M as E, σ and variables of αM as E', σ'. Then $E'', \sigma'', \alpha M = M \Downarrow 1$, so by rewriting $E'', \sigma'', (\alpha y)[\widetilde{M'}] = y[\widetilde{M}] \Downarrow 1$, so $E, \sigma, y[\widetilde{M}] \Downarrow b$.
  - Otherwise, there is no E, σ such that $E, \sigma, M \Downarrow a$, so the result holds trivially.

  Let us now consider the two cases of Rule (4).
  - First case: x occurs in M_1, x is defined by restrictions *new* $x : T$, T is a large type, M_1 characterizes x, and M_2 is obtained by optionally applying function symbols to terms of the form $y[\widetilde{M'}]$ where y is defined by restrictions and $y \neq x$.

    Since M_1 characterizes x, there exist $\widetilde{M}$ and a function f from bitstrings to bitstrings such that for all E, σ, and a, $E, \sigma, M_1 \Downarrow a$ implies $E, \sigma, x[\widetilde{M}] \Downarrow f(a)$. The set S_2 of values of M_2 for all values of indexes of variables y has cardinal polynomial in η, $q_2(\eta)$. Moreover, this set is independent of the values of x (although the values of indexes of y may depend on x). The probability that the value of M_1 is in S_2 is therefore at most the probability that $x[\widetilde{b}]$ is in $f(S_2)$ for some $\widetilde{b}$, so at most $\frac{q_1(\eta)q_2(\eta)}{|I_\eta(T)|}$ where $q_1(\eta)$ is the number of possible values of the indexes $\widetilde{b}$ of x, which is polynomial in η.

    When the value of M_1 is not in S_2, $E, \sigma, M_1 = M_2 \Downarrow 0$, so $E, \sigma \not\vdash (\mathcal{F}_{m'-1}, \mathcal{R}_{m'-1})$ so the result follows by induction hypothesis.
  - Second case: x occurs in M_1, x is defined by restrictions *new* $x : T$, T is a large type, M_1 characterizes x, only_dep$(x) = S$, and no variable of S occurs in M_2.

    We consider traces of $C[Q_0]$ that differ by the choices of values of x. Since only_dep$(x) = S$, these traces differ only by the values of variables in S, after excluding exceptional traces in which we have $E, M_1 = M_2 \Downarrow 1$ for M_1, M_2 considered in Rule (4) or for some test $M_1 = M_2$ or $M_1 \neq M_2$ in Q_0 such that there exists $y \in S$ such that M_1 characterizes y, and no variable in S occurs in M_2. Due to the form of *let* assignments required by only_dep and the properties of "characterize", there exist $\widetilde{M}$ and functions $f_1, \ldots, f_m$ from bitstrings to bitstrings such that for

all E and a, $E, M_1 \Downarrow a$ implies $E, x[\widetilde{M}] \Downarrow f_j(a)$ for some $j \leq m$. (We may need several functions because variables in S may have several definitions.) In the considered traces, the value of M_2 is the same a, and the probability that $E, M_1 \Downarrow a$ is at most the probability that $E(x[\widetilde{b}]) = f_j(a)$ for some $\widetilde{b}$ and some $j \leq m$, so at most $\frac{q_1(\eta)}{|I_\eta(T)|}$ where $q_1(\eta)$ is the number of possible values of the indexes $\widetilde{b}$ of x, which is polynomial in η. Therefore, the probability of excluded traces is at most $\frac{q_1(\eta)q_2(\eta)}{|I_\eta(T)|}$ where the number of executions of such tests $M_1 = M_2$ or $M_1 \neq M_2$ is at most $q_2(\eta)$, polynomial in η.
For traces that have not been excluded, $E, M_1 = M_2 \Downarrow 0$, so $E, \sigma \not\vdash (\mathcal{F}_{m'-1}, \mathcal{R}_{m'-1})$ so the result follows by induction hypothesis.

It is then easy to show the correctness of game simplification. For simplicity, we consider one transformation at a time, and use transitivity of $\approx$ to conclude when several transformations are applied. For each trace $\text{initConfig}(C[Q_0]) \to \ldots \to E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$, except in cases of negligible probability, we show that there exists a corresponding trace $\text{initConfig}(C[Q'_0]) \to \ldots \to E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ with $E'_{m'} = E_m$, $P'_{m'}$ is obtained from P_m by the same transformation as Q'_0 from Q_0, $\mathcal{Q}'_{m'}$ is obtained from $\mathcal{Q}_m$ by the same transformation as Q'_0 from Q_0, $\mathcal{C}'_{m'} = \mathcal{C}_m$, with the same probability. The proof proceeds by induction on m. The case $m = 0$ is obvious, since the game simplifications do not change input processes. For the inductive step, we reason by cases on the last reduction step of the trace of $C[Q_0]$. We consider only the cases in which the transition may be altered by the game simplification.

- Case 1: M reduces into M' by a user-defined rewrite rule, and we replace M with M' in the smallest process $P_M = C_M[M]$ that contains M. If $E, \sigma, M \Downarrow a$ then $E, \sigma, M' \Downarrow a'$ (since the variable accesses in M' are included in those of M and M and M' are well-typed). When $a \neq a'$, the game provides an adversary that satisfies the conditions of the definition of the corresponding user claim (as in the item "Cases (1) and (6) when the reduction uses a user-defined rewrite rule" above) so this situation has negligible probability and can be excluded. Otherwise, $a = a'$, and $C_M[M']$ reduces in the same way as $P_M = C_M[M]$.
- Case 2: M reduces into M' by a rule of $\mathcal{R}$, and we replace M with M' in the smallest process $P_M = C_M[M]$ that contains M, where $\mathcal{R}$ is the set of rewrite rules obtained by the equational prover from TrueFacts_{P_M}. We exclude traces such that $(\mathcal{F}_0, \mathcal{R}_0) = (\text{TrueFacts}_{P_M}, \emptyset) \wedge \forall j \leq m', \frac{\mathcal{F}_{j-1}, \mathcal{R}_{j-1}}{\mathcal{F}_j, \mathcal{R}_j} \wedge E, \sigma \not\vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})$. (They have negligible probability by the correctness of the equational prover.) In the remaining traces, for all $(M_1 \to M_2) \in \mathcal{R} = \mathcal{R}_{m'}$, $E, \sigma \vdash M_1 = M_2$. So $E, \sigma, M \Downarrow a$ if and only if $E, \sigma, M' \Downarrow a$, and $C_M[M']$ reduces in the same way as $P_M = C_M[M]$.
- Case 3: $P = \mathit{if}\ M\ \mathit{then}\ P_1\ \mathit{else}\ P_2$, TrueFacts_{P_2} yields a contradiction, and we replace P with P_1. We exclude traces that reduce P by (If2), yielding P_2. Indeed, $\Pr[\text{initConfig}(C[Q_0]) \to \ldots \to E, (\sigma, P_2), \mathcal{Q}, \mathcal{C}] = \Pr[\text{initConfig}(C[Q_0]) \to \ldots \to E, (\sigma, P_2), \mathcal{Q}, \mathcal{C} \wedge (\mathcal{F}_0, \mathcal{R}_0) = (\text{TrueFacts}_{P_2}, \emptyset) \wedge \forall j \leq m', \frac{\mathcal{F}_{j-1}, \mathcal{R}_{j-1}}{\mathcal{F}_j, \mathcal{R}_j} \wedge E, \sigma \not\vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})]$ since $E, \sigma \not\vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})$ is always true since TrueFacts_{P_2} yields a contradiction. So the excluded traces have negligible probability by the correctness of the equational prover. In the remaining traces, P reduces to P_1 by (If1), so replacing P with P_1 just removes one reduction step without otherwise changing the trace.
- The other cases can be handled in a similar way.

We also show the converse property: for each trace of $C[Q'_0]$, except in cases of negligible probability, there exists a corresponding trace of $C[Q_0]$ with the same probability. Moreover, for all channels c and bitstrings a, $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ executes $\overline{c}\langle a\rangle$ immediately if and only if $E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ executes $\overline{c}\langle a\rangle$ immediately, so $\Pr[C[Q_0] \rightsquigarrow_\eta \overline{c}\langle a\rangle] = \Pr[C[Q'_0] \rightsquigarrow_\eta \overline{c}\langle a\rangle]$, which yields the desired equivalence. $\square$

B.4 Proof of Proposition 3

Proof of Proposition 3 Let C be an evaluation context acceptable for $[\![L]\!]$, $[\![R]\!]$, $\emptyset$.

We define a probabilistic polynomial Turing machine $\mathcal{A}_a$, for $a \in [1, I_\eta(n'')]$, as follows. $\mathcal{A}_a$ uses oracles $mac(., k)$ and $check(., k, .)$. $\mathcal{A}_a$ simulates $C[\llbracket L \rrbracket]$ except that:

- for $a' < a$, in copies corresponding to $i'' = a'$ of L, $\mathcal{A}_a$ computes $\mathit{find}\ u \leq n\ \mathit{suchthat}\ \mathit{defined}(x[u]) \wedge (m = x[u]) \wedge \mathit{check}(m, \mathit{mkgen}(r), ma)\ \mathit{then}\ 1\ \mathit{else}\ 0$ instead of $\mathit{check}(m, \mathit{mkgen}(r), ma)$, and
- in the copy corresponding to $i'' = a$, $\mathcal{A}_a$ does not choose a random number $r[a]$, it calls the oracle $mac(., k)$ on x instead of computing $mac(x, \mathit{mkgen}(r))$, and instead of computing $\mathit{check}(m, \mathit{mkgen}(r), ma)$, it computes $b_1 = \mathit{check}(m, k, ma)$ using the oracle $\mathit{check}(., k, .)$ and $b_2 = \mathit{find}\ u \leq n\ \mathit{suchthat}\ \mathit{defined}(x[j]) \wedge (m = x[u]) \wedge b_1\ \mathit{then}\ 1\ \mathit{else}\ 0$; if $b_1 \neq b_2$, the execution of the Turing machine stops, with result (m, ma); otherwise, the execution continues using value $b_1 = b_2$.

When $\mathcal{A}_a$ has not stopped due to the last item above, it returns $\perp$ when the simulation of $C[\llbracket L \rrbracket]$ terminates.

When $\mathcal{A}_a$ returns (m, t), $b_1 \neq b_2$. Moreover, if $b_1 = 0$, then $b_2 = 0$ by definition of b_2. So $b_1 = 1$ and $b_2 = 0$. Therefore, there is no j such that $m = x[j]$, hence $\mathcal{A}_a$ has not called the oracle $mac(., k)$ on m. Moreover, there exists a polynomial q such that for all a, $\mathcal{A}_a$ runs in time $q(\eta)$. So by Definition 1, $\max_a p_a(\eta)$ is negligible, where

$$p_a(\eta) = \Pr[r \overset{R}{\leftarrow} I_\eta(T_{mr}); k \leftarrow \mathit{mkgen}_\eta(r); (m, t) \leftarrow \mathcal{A}_a : \mathit{check}_\eta(m, k, t)]$$

Since $I_\eta(n'')$ is polynomial in η, $\sum_{a \in [1, I_\eta(n'')]} p_a(\eta) \leq \max_a p_a(\eta) \times I_\eta(n'')$ is also negligible.

On the other hand, let c be a channel and a' be a bitstring. We need to evaluate $|\Pr[C[\llbracket L \rrbracket] \rightsquigarrow_\eta \overline{c}\langle a' \rangle] - \Pr[C[\llbracket R \rrbracket] \rightsquigarrow_\eta \overline{c}\langle a' \rangle]|$. We consider three categories of pairs of traces $(\mathcal{T}, \mathcal{T}')$ where $\mathcal{T}$ and $\mathcal{T}'$ are traces of $C[\llbracket L \rrbracket$ and $C[\llbracket R \rrbracket]$ respectively:

1. Traces $\mathcal{T}$ and $\mathcal{T}'$ have the same configurations except for the replacement of L with R in processes, they terminate, and none of their configurations executes $\overline{c}\langle a' \rangle$ immediately.
2. Traces $\mathcal{T}$ and $\mathcal{T}'$ have the same configurations except for the replacement of L with R in processes up to a point at which their corresponding configurations both execute $\overline{c}\langle a' \rangle$ immediately.
3. Traces $\mathcal{T}$ and $\mathcal{T}'$ have the same configurations except for the replacement of L with R in processes up to a point at which their configurations differ because for some $a \in [1, I_\eta(n'')]$, for some messages m, ma received on channel $c_2[a]$ (where c_2 is the channel used in $\llbracket L \rrbracket$ and $\llbracket R \rrbracket$ for the second parallel process of L and R), the result returned by $\llbracket L \rrbracket$ differs from the one returned by $\llbracket R \rrbracket$. In this case, the simulating Turing machine that runs $r \overset{R}{\leftarrow} I_\eta(T_{mr}); k \leftarrow \mathit{mkgen}_\eta(r)$ and executes $\mathcal{A}_a$ will return (m, ma), by construction.

All traces of $C[\llbracket L \rrbracket]$ fall in one of the above categories, and similarly for traces of $C[\llbracket R \rrbracket]$. Traces of the first category have no contribution to $\Pr[C[\llbracket L \rrbracket] \rightsquigarrow_\eta \overline{c}\langle a' \rangle]$ and to $\Pr[C[\llbracket R \rrbracket] \rightsquigarrow_\eta \overline{c}\langle a' \rangle]$; traces of the second category cancel out when computing $\Pr[C[\llbracket L \rrbracket] \rightsquigarrow_\eta \overline{c}\langle a' \rangle] - \Pr[C[\llbracket R \rrbracket] \rightsquigarrow_\eta \overline{c}\langle a' \rangle]$. So $|\Pr[C[\llbracket L \rrbracket] \rightsquigarrow_\eta \overline{c}\langle a' \rangle] - \Pr[C[\llbracket R \rrbracket] \rightsquigarrow_\eta \overline{c}\langle a' \rangle]| \leq \Pr[(\mathcal{T}, \mathcal{T}')$ is in the third category$] \leq \sum_{a \in [1, I_\eta(n'')]} \Pr[r \overset{R}{\leftarrow} I_\eta(T_{mr}); k \leftarrow \mathit{mkgen}_\eta(r); (m, t) \leftarrow \mathcal{A}_a] = \sum_{a \in [1, I_\eta(n'')]} p_a(\eta)$.

Hence $|\Pr[C[\llbracket L \rrbracket] \rightsquigarrow_\eta \overline{c}\langle a' \rangle] - \Pr[C[\llbracket R \rrbracket] \rightsquigarrow_\eta \overline{c}\langle a' \rangle]|$ is negligible, so $\llbracket L \rrbracket \approx \llbracket R \rrbracket$. □

B.5 Proof of Proposition 4

Let us first introduce some notations. We denote by $L_{j_0, \ldots, j_k}$ the subtrees of L defined as follows:

$$L = (L_1, \ldots, L_{m'})$$
$$L_{j_0, \ldots, j_k} = !^{i \leq n} \mathit{new}\ y_1 : T_1; \ldots; \mathit{new}\ y_m : T_m; (L_{j_0, \ldots, j_k, 1}, \ldots, L_{j_0, \ldots, j_k, m'})$$

Then we define $i_{j_0,\dots,j_k} = i$, $n_{j_0,\dots,j_k} = n$, $y_{(j_0,\dots,j_k),k'} = y_{k'}$, and $\mathrm{nNew}_{j_0,\dots,j_k} = m$. When $L_{j_0,\dots,j_l} = (x_1 : T_1, \dots, x_m : T_m) \rightarrow FP$, we say that $L_{j_0,\dots,j_l}$ is a leaf of L, and we define $x_{(j_0,\dots,j_l),k'} = x_{k'}$, $T_{(j_0,\dots,j_l),k'} = T_{k'}$, and $\mathrm{nInput}_{j_0,\dots,j_l} = m$.

In order to prove Proposition 4, we define a context C such that $Q_0 \approx_0^V C[\![L]\!]$ and $C[\![R]\!] \approx_0^V Q_0'$. We first define a process $\mathrm{relay}(L)$ as follows:

$$\mathrm{relay}((G_1, \dots, G_m)) = \mathrm{relay}(G_1)^1 \mid \dots \mid \mathrm{relay}(G_m)^m$$

$$\mathrm{relay}(!^{i \le n} \mathit{new}\ y_1 : T_1; \dots; \mathit{new}\ y_l : T_l; (G_1, \dots, G_m))^{\widetilde{j}}_{\widetilde{i}} =$$

$$!^{i \le n} d_{\widetilde{j}}[\widetilde{i}, i](); \overline{c_{\widetilde{j}}[\widetilde{i}, i]}\langle\rangle; c_{\widetilde{j}}[\widetilde{i}, i](); \overline{d_{\widetilde{j}}[\widetilde{i}, i]}\langle\rangle; (\mathrm{relay}(G_1)^{\widetilde{j},1}_{\widetilde{i},i} \mid \dots \mid \mathrm{relay}(G_m)^{\widetilde{j},m}_{\widetilde{i},i} \mid !^{i' \le n'} d_{\widetilde{j}}[\widetilde{i}, i](); \overline{d_{\widetilde{j}}[\widetilde{i}, i]}\langle\rangle)$$

$$\mathrm{relay}((x_1 : T_1, \dots, x_l : T_l) \rightarrow FP)^{\widetilde{j}}_{\widetilde{i}} = d_{\widetilde{j}}[\widetilde{i}](x_1 : T_1, \dots, x_l : T_l); \overline{c_{\widetilde{j}}[\widetilde{i}]}\langle x_1, \dots, x_l\rangle; c_{\widetilde{j}}[\widetilde{i}](r : \mathit{bitstring});$$

$$\overline{d_{\widetilde{j}}[\widetilde{i}]}\langle r\rangle; !^{i' \le n'} d_{\widetilde{j}}[\widetilde{i}](x_1 : T_1, \dots, x_l : T_l); \overline{d_{\widetilde{j}}[\widetilde{i}]}\langle r\rangle$$

where $\widetilde{i} = i_1, \dots, i_{l'}$ and $\widetilde{j} = j_0, \dots, j_{l'}$. This relay process relays messages sent on channel $d_{\widetilde{j}}$ to channel $c_{\widetilde{j}}$ so that the corresponding random numbers $y_1, \dots, y_l$ are chosen. When those random numbers have already been chosen, the process accepts messages on $d_{\widetilde{j}}$ but yields control back to the sending process without executing anything by outputting on $d_{\widetilde{j}}$. The relay process also allows calling several times the same functional process FP with the same values of $\widetilde{j}$ and $\widetilde{i}$, in which case it always returns the same result r. (We make sure in the following that when a functional process is called several times, the calls all use the same arguments.) Since L and R are required to have the same structure by Hypothesis H2, $\mathrm{relay}(L) = \mathrm{relay}(R)$.

We introduce the following auxiliary definitions, which allow us to define the correspondence mapIdx_M from replication indexes at M in Q_0 to replication indexes at N_M in L:

- For each $M \in \mathcal{M}$ and $k \le \mathrm{nNewSeq}_M$, we define $\mathrm{count}_\eta(k, M)$ as follows. Let $n_1, \dots, n_l$ be the sequence of bounds of replications above the definition of $z_{kk',M}$ for any k'. Let l' be the length of the longest common prefix of $\mathrm{im\ index}_k(M)$ and $\mathrm{im\ index}_{k_0}(M)$ for $k_0 < k$. We define $\mathrm{count}_\eta(k, M) = I_\eta(n_{l'+1}) \times \dots \times I_\eta(n_l)$.
  We define parameters $\mathrm{count}_{k,M}$ such that $I_\eta(\mathrm{count}_{k,M}) = \mathrm{count}_\eta(k, M)$.
  We define function symbols $\mathrm{num}_{k,M} : [1, n_1] \times \dots \times [1, n_l] \rightarrow [1, \mathrm{count}_{k,M}]$ such that $I_\eta(\mathrm{num}_{k,M})(a_1, \dots, a_l) = 1 + (a_{l'+1} - 1) + I_\eta(n_{l'+1}) \times ((a_{l'+2} - 1) + I_\eta(n_{l'+2}) \times \dots + I_\eta(n_{l-1}) \times (a_l - 1))$. Then $\mathrm{num}_{k,M}$ establishes a bijection between the last $l - l'$ components of its argument and its result.
- We define $\mathrm{tot\_count}_\eta(j_0, \dots, j_k)$ as the sum of $\mathrm{count}_\eta(k+1, M'')$ for all M'' such that the first $k+1$ elements of $BL(M'')$ are equal to $j_0, \dots, j_k$, counting only once terms M'' that share the first $k+1$ sequences of random variables.
  We set $I_\eta(n_{j_0,\dots,j_k}) = \mathrm{tot\_count}_\eta(j_0, \dots, j_k)$, where $n_{j_0,\dots,j_k}$ is the bound of some replication in L. $I_\eta(n_{j_0,\dots,j_k})$ is then large enough so that there is always an available copy of the desired replicated process when we need to execute one.
  The replication at the root of $\mathrm{relay}(L_{j_0,\dots,j_k})^{j_0,\dots,j_k}_{i_1,\dots,i_k}$ is also bounded by $n_{j_0,\dots,j_k}$. The other replication of $\mathrm{relay}(L_{j_0,\dots,j_k})^{j_0,\dots,j_k}_{i_1,\dots,i_k}$ is bounded by n', where $I_\eta(n')$ is the sum for all $M \in \mathcal{M}$ of $I_\eta(n_1) \times \dots \times I_\eta(n_l)$ where $n_1, \dots, n_l$ is the sequence of bounds of replications above M in Q_0.
- We order the term occurrences in $\mathcal{M}$ arbitrarily, with a total ordering. Let $\mathrm{start}_\eta(k, M)$ be defined as follows. Let M' the smallest term occurrence of $\mathcal{M}$ that shares the first k sequences of random variables with M. Then $\mathrm{start}_\eta(k, M)$ is the sum of $\mathrm{count}_\eta(k, M'')$ for all M'' smaller than M' such that the first k elements of $BL(M'')$ are equal to the first k elements of $BL(M')$, counting only once terms M'' that share the first k sequences of random variables.
  We define function symbols $\mathrm{addstart}_{k,M} : [1, \mathrm{count}_{k,M}] \rightarrow [1, n_{j_0,\dots,j_k}]$ where $BL(M) = (j_0, \dots, j_k, \dots)$, such that $I_\eta(\mathrm{addstart}_{k,M})(a) = \mathrm{start}_\eta(k, M) + a$.

– We define the sequence of terms $\mathrm{convindex}(k, M) = (\mathrm{addstart}_{1,M}(\mathrm{num}_{1,M}(\mathrm{im}\ \mathrm{index}_1(M))), \dots, \mathrm{addstart}_{k,M}(\mathrm{num}_{k,M}(\mathrm{im}\ \mathrm{index}_k(M))))$.
This sequence of terms implements the function $mapIdx_M$ mentioned in the explanation of the transformation, in Section 3.3. More precisely, $mapIdx_M(\widetilde{a}) = \mathrm{convindex}(l, M)\{\widetilde{a}/\widetilde{i}\}$, where $\widetilde{i}$ is the sequence of current replication indexes at M and $l = \mathrm{nNewSeq}_M$.

Then we define $C = (newChannel\ c_{\widetilde{j}}; newChannel\ d_{\widetilde{j}};)_{\widetilde{j}}([\,] \mid \mathrm{relay}(L) \mid Q_0''))$ where the process Q_0'' is defined from Q_0 as follows:

– When $x \in S$, we replace its definition $new\ x : T; Q$ with $let\ x : T = cst\ in\ Q$ for some constant cst.
– For each $M \in \mathcal{M}$, let $P_M = C_M[M]$ be the smallest subprocess of Q_0 containing M. Let $l = \mathrm{nNewSeq}_M$ and $m = \mathrm{nInput}_M$. Let $BL(M) = (j_0, \dots, j_l)$. Let $d_M = d_{j_0,\dots,j_l}[\mathrm{convindex}(l, M)]$ and for all $k \leq l$, $d_{M,k} = d_{j_0,\dots,j_{k-1}}[\mathrm{convindex}(k, M)]$. We replace P_M with $\overline{d_{M,1}}\langle\rangle; d_{M,1}(); \dots \overline{d_{M,l}}\langle\rangle; d_{M,l}();$ $\overline{d_M}\langle\sigma_M x_{1,M}, \dots, \sigma_M x_{m,M}\rangle; d_M(y : bitstring); C_M[y]$ where y is a fresh variable.

Lemma 11. $Q_0 \approx_0^V C[[\![L]\!]]$

Proof. The bounds of replications of $[\![L]\!]$ and $\mathrm{relay}(L)$ have been defined above. As outlined in the proof of Property P4, the length of all bitstrings manipulated by Q_0 is polynomial in η. We can therefore define $\mathrm{maxlen}_\eta(c_{\widetilde{j}})$ to be a polynomial large enough so that messages sent on $c_{\widetilde{j}}$ by $C[[\![L]\!]]$ are never truncated. We define $\mathrm{maxlen}_\eta(d_{\widetilde{j}}) = \mathrm{maxlen}_\eta(c_{\widetilde{j}})$, then messages on $d_{\widetilde{j}}$ are never truncated.

Let C' be any evaluation context acceptable for Q_0, $C[[\![L]\!]]$, V. We relate traces of $C'[Q_0]$ and of $C'[C[[\![L]\!]]]$ as follows.

We assume that the channels $c_{\widetilde{j}}$ and $d_{\widetilde{j}}$ do not occur in C' and Q_0, and that during reductions (NewChannel), these channels are substituted by themselves. (This is easy to guarantee by renaming; this assumption simplifies notations in the proof.)

We write $M =_E M'$ when $E, M \Downarrow a$ and $E, M' \Downarrow a$ for some bitstring a. We denote by k-th$(\widetilde{i})$ the k-th component of the tuple $\widetilde{i}$, and by $|\widetilde{i}|$ the number of elements of the tuple $\widetilde{i}$.

We define a relation between variables of S in Q_0 and variables y defined by new in $[\![L]\!]$: we say that $y[a_1, \dots, a_j] \xrightarrow{\mathrm{var}}_E \mathrm{varImL}(y, M)[\widetilde{a'}]$ when for all $k' \leq j$, $E, \mathrm{addstart}_{k',M}(\mathrm{num}_{k',M}(\mathrm{im}\ (\rho_{j-1}(M) \circ \dots \circ \rho_{k'}(M))\{\widetilde{a'}/\widetilde{i}\})) \Downarrow a_{k'}$, where $\widetilde{i} \leq \widetilde{n}$ are the current replication indexes at the definition of $\mathrm{varImL}(y, M)$ with their associated bounds, and for all $l \leq |\widetilde{i}|$, l-th$(\widetilde{a'}) \in [1, I_\eta(l\text{-th}(\widetilde{n}))]$. (Note that $\xrightarrow{\mathrm{var}}$ depends on η.)

We show that the relation $\xrightarrow{\mathrm{var}}_E$ is a (partial) function, that is, if $y[a_1, \dots, a_j] \xrightarrow{\mathrm{var}}_E M_V$ and $y[a_1, \dots, a_j] \xrightarrow{\mathrm{var}}_E M'_V$ then $M_V = M'_V$. Assume that $y[a_1, \dots, a_j] \xrightarrow{\mathrm{var}}_E z'[\widetilde{a'}]$ and $y[a_1, \dots, a_j] \xrightarrow{\mathrm{var}}_E z''[\widetilde{a''}]$. Then

– $z' = \mathrm{varImL}(y, M')$, $\ E, \mathrm{addstart}_{k',M'}(\mathrm{num}_{k',M'}(\mathrm{im}\ (\rho_{j-1}(M') \circ \dots \circ \rho_{k'}(M'))\{\widetilde{a'}/\widetilde{i'}\})) \Downarrow a_{k'}$ for all $k' \leq j$, where $\widetilde{i'} \leq \widetilde{n'}$ are the current replication indexes at the definition of z' with their associated bounds, and for all $l \leq |\widetilde{i'}|$, l-th$(\widetilde{a'}) \in [1, I_\eta(l\text{-th}(\widetilde{n'}))]$,
– $z'' = \mathrm{varImL}(y, M'')$, $\ E, \mathrm{addstart}_{k',M''}(\mathrm{num}_{k',M''}(\mathrm{im}\ (\rho_{j-1}(M'') \circ \dots \circ \rho_{k'}(M''))\{\widetilde{a''}/\widetilde{i''}\})) \Downarrow a_{k'}$ for all $k' \leq j$, where $\widetilde{i''} \leq \widetilde{n''}$ are the current replication indexes at the definition of z'' with their associated bounds, and for all $l \leq |\widetilde{i''}|$, l-th$(\widetilde{a''}) \in [1, I_\eta(l\text{-th}(\widetilde{n''}))]$.

For all terms M'', we have either $\mathrm{start}_\eta(k', M'') \leq \mathrm{start}_\eta(k', M')$ or $\mathrm{start}_\eta(k', M'') \geq \mathrm{start}_\eta(k', M') + \mathrm{count}_\eta(k', M')$ since $\mathrm{start}_\eta(k', M'')$ is computed by adding $\mathrm{count}_\eta(k', M_3)$ for some terms M_3 in a fixed order. Moreover, $\mathrm{num}_{k',M'}(\dots)$ evaluates to a bitstring in $[1, \mathrm{count}_\eta(k', M')]$. Therefore, $\mathrm{start}_\eta(k', M'') \leq \mathrm{start}_\eta(k', M')$. By symmetry, $\mathrm{start}_\eta(k', M'') \geq \mathrm{start}_\eta(k', M')$. So we have for all $k' \leq j$, $\mathrm{start}_\eta(k', M') = \mathrm{start}_\eta(k', M'')$ and $\mathrm{num}_{k',M'}(\mathrm{im}\ (\rho_{j-1}(M') \circ \dots \circ \rho_{k'}(M'))\{\widetilde{a'}/\widetilde{i'}\}) =_E \mathrm{num}_{k',M''}(\mathrm{im}\ (\rho_{j-1}(M'') \circ \dots \circ \rho_{k'}(M''))\{\widetilde{a''}/\widetilde{i''}\})$. Since $\mathrm{start}_\eta(j, M') = \mathrm{start}_\eta(j, M'')$, by definition of start_η, M' shares the first j

sequences of random variables with M''. Since y has j indexes, y is defined under j replications in L, so $\mathrm{varImL}(y, M') = \mathrm{varImL}(y, M'')$, that is, $z' = z''$. So $j' = j''$. By Hypothesis H$'$4.2, $\rho_{k'}(M') = \rho_{k'}(M'')$ for all $k' < j$. By definition of num, $I_\eta(\mathrm{num}_{k',M'}) = I_\eta(\mathrm{num}_{k',M''})$ for all $k' \leq j$.

We show by induction on k' that if for all $k'' \leq k'$, $\mathrm{num}_{k'',M'}(\mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a'}/\widetilde{i'}\}) =_E \mathrm{num}_{k'',M'}(\mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a''}/\widetilde{i'}\})$, where $\widetilde{i'} \leq \widetilde{n'}$ are the current replication indexes at the definition of $z_{k'\_,M'}$ with their associated bounds, and $l\text{-th}(\widetilde{a'}), l\text{-th}(\widetilde{a''}) \in [1, I_\eta(l\text{-th}(\widetilde{n'}))]$, then $\widetilde{a'} = \widetilde{a''}$.

- For $k' = 1$, we assume $\mathrm{num}_{1,M'}(\widetilde{a'}) =_E \mathrm{num}_{1,M'}(\widetilde{a''})$. The longest common prefix of $\mathrm{index}_1(M')$ and $\mathrm{index}_{j''}(M')$ for $j'' < 1$ is empty, since $\mathrm{index}_{j''}(M')$ is defined only for $j'' \geq 1$. So $\mathrm{num}_{1,M'}$ establishes a bijection between the tuples $\widetilde{a'}$ smaller than the current replication bounds at definition of $z_{1\_,M'}$ and the interval $[1, \mathrm{count}_\eta(1, M')]$. So $\widetilde{a'} = \widetilde{a''}$.
- Assume that $\mathrm{num}_{k'',M'}(\mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a'}/\widetilde{i'}\}) =_E \mathrm{num}_{k'',M'}(\mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a''}/\widetilde{i'}\})$ for all $k'' \leq k'$. Let $k'_{\mathrm{ind}} < k'$. Let $E, \mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k'_{\mathrm{ind}}}(M'))\{\widetilde{a'}/\widetilde{i'}\} \Downarrow \widetilde{a'}_{\mathrm{ind}}$ and $E, \mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k'_{\mathrm{ind}}}(M'))\{\widetilde{a''}/\widetilde{i'}\} \Downarrow \widetilde{a''}_{\mathrm{ind}}$. By hypothesis, we have for all $k'' \leq k'_{\mathrm{ind}}$, $\mathrm{num}_{k'',M'}(\mathrm{im}\ (\rho_{k'_{\mathrm{ind}}-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a'}_{\mathrm{ind}}/\widetilde{i'}_{\mathrm{ind}}\}) =_E \mathrm{num}_{k'',M'}(\mathrm{im}\ (\rho_{k'_{\mathrm{ind}}-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a''}_{\mathrm{ind}}/\widetilde{i'}_{\mathrm{ind}}\})$ where $\widetilde{i'}_{\mathrm{ind}} \leq \widetilde{n'}_{\mathrm{ind}}$ are the current replication indexes at the definition of $z_{k'_{\mathrm{ind}}\_,M'}$ with their associated bounds. By induction hypothesis, $\widetilde{a'}_{\mathrm{ind}} = \widetilde{a''}_{\mathrm{ind}}$, so for all $k'' < k'$, $\mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a'}/\widetilde{i'}\} =_E \mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a''}/\widetilde{i'}\}$. For $k'' = k'$, we have $\mathrm{num}_{k',M'}(\widetilde{a'}) =_E \mathrm{num}_{k',M'}(\widetilde{a''})$.
  Let l be the length of the longest common prefix of $\mathrm{im}\ \mathrm{index}_{k'}(M')$ and $\mathrm{im}\ \mathrm{index}_{k''_0}(M')$ for $k''_0 < k'$. Since $\mathrm{index}_{k''_0}(M') = \mathrm{index}_{k'}(M') \circ \rho_{k'-1}(M') \circ \ldots \circ \rho_{k''_0}(M')$, the first l components of $\mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''_0}(M'))$ are then the first l components of $\widetilde{i'}$, so the first l components of $\widetilde{a'}$ and $\widetilde{a''}$ are equal. Moreover $\mathrm{num}_{k',M'}$ establishes a bijection between the last $|\widetilde{a'}| - l$ components of its argument and the interval $[1, \mathrm{count}_\eta(k', M')]$. So the last $|\widetilde{a'}| - l$ components of $\widetilde{a'}$ and $\widetilde{a''}$ are equal. Hence $\widetilde{a'} = \widetilde{a''}$.

Therefore, we conclude that $\widetilde{a'} = \widetilde{a''}$, so $z'[\widetilde{a'}] = z''[\widetilde{a''}]$.

Next, we show that the function $\xrightarrow{\mathrm{var}}_E$ is injective. If $y'[a'_1, \ldots, a'_{j'}] \xrightarrow{\mathrm{var}}_E z[a_1, \ldots, a_j]$ and $y''[a''_1, \ldots, a''_{j''}] \xrightarrow{\mathrm{var}}_E z[a_1, \ldots, a_j]$, then $z = \mathrm{varImL}(y', M')$ and $z = \mathrm{varImL}(y'', M'')$. By Hypothesis H$'$4.1, M' and M'' share at least the first $j' = j''$ sequences of random variables and $y' = y''$. By Hypothesis H$'$4.2, $\rho_{k'}(M') = \rho_{k'}(M'')$ for all $k' < j' = j''$. By definition of addstart and num, $\mathrm{start}_\eta(k', M') = \mathrm{start}_\eta(k', M'')$ and $I_\eta(\mathrm{num}_{k',M'}) = I_\eta(\mathrm{num}_{k',M''})$ for all $k' \leq j' = j''$. Hence $a'_{k'} = a''_{k'}$ for all $k' \leq j' = j''$. So $y'[a'_1, \ldots, a'_{j'}] = y''[a''_1, \ldots, a''_{j''}]$.

For each trace $\mathrm{initConfig}(C'[Q_0]) \to \ldots \to E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ of $C'[Q_0]$ of probability p_m, we show that there exists a trace $\mathrm{initConfig}(C'[C[\llbracket L \rrbracket]]) \to \ldots \to E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ of $C'[C[\llbracket L \rrbracket]]$ of probability $p'_{m'}$ such that

- For all $z \notin S$, $E'_{m'}(z[a'_1, \ldots, a'_{j'}]) = E_m(z[a'_1, \ldots, a'_{j'}])$; for all $z \in S$, $z[a'_1, \ldots, a'_{j'}]$ is in $\mathrm{Dom}(E_m)$ if and only if it is in $\mathrm{Dom}(E'_{m'})$; if y is defined by *new* in L and $y[a_1, \ldots, a_j] \in \mathrm{Dom}(E'_{m'})$ then there exists M_V such that $y[a_1, \ldots, a_k] \xrightarrow{\mathrm{var}}_{E_m} M_V$ and $M_V \in \mathrm{Dom}(E_m)$ and for all such M_V, $E'_{m'}(y[a_1, \ldots, a_j]) = E_m(M_V)$.
- $P'_{m'}$ is obtained from P_m as Q''_0 from Q_0 (transforming only the occurrences that appear in P_m), $\mathcal{Q}'_{m'} = \mathcal{Q}^1_{m'} \uplus \mathcal{Q}^2_{m'} \uplus \mathcal{Q}^3_{m'}$, where $\mathcal{Q}^1_{m'}$ is obtained from $\mathcal{Q}_m$ as Q''_0 from Q_0 (transforming only the occurrences that appear in $\mathcal{Q}_m$), $\mathcal{Q}^2_{m'}$ is what remains of $\mathrm{relay}(L)$ after partial execution, and $\mathcal{Q}^3_{m'}$ is what remains of $\llbracket L \rrbracket$ after partial execution. More precisely, let $\mathrm{relay}(L^{a_1,\ldots,a_k}_{j_0,\ldots,j_k}) = \mathrm{relay}(L_{j_0,\ldots,j_k})^{j_0,\ldots,j_k}_{i_1,\ldots,i_k}\{a_1/i_1, \ldots, a_k/i_k\}$ and $\llbracket L^{a_1,\ldots,a_k}_{j_0,\ldots,j_k} \rrbracket = \llbracket L_{j_0,\ldots,j_k} \rrbracket^{j_0,\ldots,j_k}_{i_1,\ldots,i_k}\{a_1/i_1, \ldots, a_k/i_k\}$ where i_1, $\ldots, i_k$ are the replications indexes of L above $L_{j_0,\ldots,j_k}$. These processes correspond respectively to the relay process and to the translation of the subtree $L_{j_0,\ldots,j_k}$ of L, for the value of the replication indexes $a_1, \ldots, a_k$. Let $\mathrm{redRepl}(a, !^{i \leq n} P) = P\{a/i\}$. Then $\mathcal{Q}^2_{m'}$ and $\mathcal{Q}^3_{m'}$ are formed as follows:

- for each $j_0, \dots, j_{k-1}$, $a_1, \dots, a_k$ such that $y_{(j_0,\dots,j_{k-1}),k'}[a_1, \dots, a_k] \in \mathrm{Dom}(E'_{m'})$, $\mathcal{Q}^2_{m'}$ contains $d_{j_0,\dots,j_{k-1}}[a_1, \dots, a_k]();\overline{d_{j_0,\dots,j_{k-1}}[a_1, \dots, a_k]}\langle\rangle$, possibly several times.
- for each $j_0, \dots, j_{k-1}$, $a_1, \dots, a_k$ such that $y_{(j_0,\dots,j_{k-2}),k''}[a_1, \dots, a_{k-1}] \in \mathrm{Dom}(E'_{m'})$ and $y_{(j_0,\dots,j_{k-1}),k'}[a_1, \dots, a_k] \notin \mathrm{Dom}(E'_{m'})$, $\mathcal{Q}^2_{m'}$ contains $\mathrm{redRepl}(a_k, \mathrm{relay}(L^{a_1,\dots,a_{k-1}}_{j_0,\dots,j_{k-1}}))$ and $\mathcal{Q}^3_{m'}$ contains $\mathrm{redRepl}(a_k, [\![L^{a_1,\dots,a_{k-1}}_{j_0,\dots,j_{k-1}}]\!])$.
- for each $j_0, \dots, j_l$, $a_1, \dots, a_l$ such that $y_{(j_0,\dots,j_{l-1}),k'}[a_1, \dots, a_l] \in \mathrm{Dom}(E'_{m'})$ and $L_{j_0,\dots,j_l}$ is a leaf of L, either $\mathcal{Q}^2_{m'}$ contains $\mathrm{relay}(L^{a_1,\dots,a_l}_{j_0,\dots,j_l})$ and $\mathcal{Q}^3_{m'}$ contains $[\![L^{a_1,\dots,a_l}_{j_0,\dots,j_l}]\!]$, or $\mathcal{Q}^2_{m'}$ contains $d_{j_0,\dots,j_l}[a_1, \dots, a_l](x_{(j_0,\dots,j_l),1} : T_{(j_0,\dots,j_l),1}, \dots, x_{(j_0,\dots,j_l),l'} : T_{(j_0,\dots,j_l),l'});\overline{d_{j_0,\dots,j_l}[a_1, \dots, a_l]}\langle r\rangle$ with $l' = \mathrm{nInput}_{j_0,\dots,j_l}$, possibly several times, and there exist $M' \in \mathcal{M}$ and $\widetilde{a'}$ such that $E_m, M'\{\widetilde{a'}/\widetilde{i'}\} \Downarrow r$, $E_m, \mathrm{convindex}(l, M')\{\widetilde{a'}/\widetilde{i'}\} \Downarrow a_1, \dots, a_l$, and $BL(M') = (j_0, \dots, j_l)$, where $\widetilde{i'}$ is the sequence of replication indexes at M'.

where for each k, a_k is a bitstring in $[1, \mathrm{tot\_count}_\eta(j_0, \dots, j_{k-1})]$.

– $\mathcal{C}'_{m'} = \mathcal{C}_m \cup \{c_{\widetilde{j}}, d_{\widetilde{j}} \mid \widetilde{j}\}$.

– $p'_{m'} = p_m \times \prod_{z,a'_1,\dots,a'_{j'}} |I_\eta(T)|$ where T is the type of z and $z \in S, a'_1, \dots, a'_{j'}$ are such that $z[a'_1, \dots, a'_{j'}] \in \mathrm{Dom}(E_m)$ and there exists no $y[a_1, \dots, a_j] \in \mathrm{Dom}(E'_{m'})$ such that $y[a_1, \dots, a_j] \xrightarrow{\mathrm{var}}_{E_m} z[a'_1, \dots, a'_{j'}]$.

Note that the same trace of $C'[C[[\![L]\!]]]$ corresponds to $\prod_{z,a'_1,\dots,a'_{j'}} |I_\eta(T)|$ traces of $C'[Q_0]$ that differ only by the values of $E_m(z[a'_1, \dots, a'_{j'}])$ for $z \in S, a'_1, \dots, a'_{j'}$ as defined in the last item above.

The proof proceeds by induction on the length m of the trace of $C'[Q_0]$. For the induction step, we distinguish cases depending on the last reduction step of the trace.

– For the initial case, we show by induction on C'' that for all $C'', \mathcal{Q}, \mathcal{C}, \sigma$ such that σ substitutes channel names for channel names without touching $c_{\widetilde{j}}$ and $d_{\widetilde{j}}$, there exist $\mathcal{Q}', \mathcal{C}', \sigma'$ such that σ' substitutes channel names for channel names without touching $c_{\widetilde{j}}$ and $d_{\widetilde{j}}$, $\emptyset, \{C''[\sigma Q_0]\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow^* \emptyset, \{\sigma' Q_0\} \uplus \mathcal{Q}', \mathcal{C}'$, and $\emptyset, \{C''[\sigma C[[\![L]\!]]]\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow^* \emptyset, \{\sigma' C[[\![L]\!]]\} \uplus \mathcal{Q}', \mathcal{C}'$. This is obvious when $C'' = [\,]$, with $\sigma' = \sigma$, $\mathcal{Q}' = \mathcal{Q}$, and $\mathcal{C}' = \mathcal{C}$. We show this result by applying (Par) when $C'' = C_1 \mid Q_1$ or $C'' = Q_1 \mid C_1$, and (NewChannel) when $C'' = \mathit{newChannel}\ c; C_1$.
So we can apply this result to $C'' = C'$, $\sigma = \mathrm{Id}$, $\mathcal{Q} = \emptyset$, and $\mathcal{C} = \mathrm{fc}(C'[Q_0])$. We have $\mathrm{fc}(C'[Q_0]) = \mathrm{fc}(C'[C[[\![L]\!]]])$, since $\mathrm{fc}(Q_0) = \mathrm{fc}(Q''_0) = \mathrm{fc}(C[[\![L]\!]])$. Therefore, there exist $\mathcal{Q}$, $\mathcal{C}$, σ such that σ substitutes channel names for channel names without touching $c_{\widetilde{j}}$ and $d_{\widetilde{j}}$, $\emptyset, \{C'[Q_0]\}, \mathrm{fc}(C'[Q_0]) \rightsquigarrow^* \emptyset, \{\sigma Q_0\} \uplus \mathcal{Q}, \mathcal{C}$, and

$$\begin{aligned}
&\emptyset, \{C'[C[[\![L]\!]]]\}, \mathrm{fc}(C'[C[[\![L]\!]]]) \rightsquigarrow^* \emptyset, \{\sigma C[[\![L]\!]]\} \uplus \mathcal{Q}, \mathcal{C} \\
&\quad \rightsquigarrow^* \emptyset, \{\sigma Q''_0, \mathrm{relay}(L), [\![L]\!]\} \uplus \mathcal{Q}, \mathcal{C} \cup \{c_{\widetilde{j}}, d_{\widetilde{j}} \mid \widetilde{j}\} && \text{by (NewChannel) and (Par)} \\
&\quad \rightsquigarrow^* \emptyset, \{\sigma Q''_0\} \uplus \mathcal{Q}^2_0 \uplus \mathcal{Q}^3_0 \uplus \mathcal{Q}, \mathcal{C} \cup \{c_{\widetilde{j}}, d_{\widetilde{j}} \mid \widetilde{j}\} && \text{by (Par) and (Repl)}
\end{aligned}$$

where $\mathcal{Q}^2_0 = \{\mathrm{redRepl}(a, \mathrm{relay}(L_{j_0})^{j_0}) \mid j_0, a \in [1, \mathrm{tot\_count}_\eta(j_0)]\}$ is what remains from $\mathrm{relay}(L)$ after expansion of parallel compositions and replications and $\mathcal{Q}^3_0 = \{\mathrm{redRepl}(a, [\![L_{j_0}]\!]^{j_0}) \mid j_0, a \in [1, \mathrm{tot\_count}_\eta(j_0)]\}$ is what remains of $[\![L]\!]$ after expansion of parallel compositions and replications.
Moreover, $\sigma Q''_0$ is obtained from σQ_0 as Q''_0 from Q_0, and $\mathcal{Q}$ does not contain any occurrence modified when transforming Q_0 into Q''_0, so $\{\sigma Q''_0\} \uplus \mathcal{Q}$ is obtained from $\{\sigma Q_0\} \uplus \mathcal{Q}$ as Q''_0 from Q_0.
Reducing $\{\sigma Q''_0\} \uplus \mathcal{Q}$ and $\{\sigma Q_0\} \uplus \mathcal{Q}$ by $\rightsquigarrow$ until they are in normal form, we obtain that $\mathrm{reduce}(\emptyset, \{C'[Q_0]\}, \mathrm{fc}(C'[Q_0])) = (\emptyset, \mathcal{Q}_0, \mathcal{C}')$ and $\mathrm{reduce}(\emptyset, \{C'[C[[\![L]\!]]]\}, \mathrm{fc}(C'[C[[\![L]\!]]])) = (\emptyset, \mathcal{Q}^1_0 \uplus \mathcal{Q}^2_0 \uplus \mathcal{Q}^3_0, \mathcal{C}' \cup \{c_{\widetilde{j}}, d_{\widetilde{j}} \mid \widetilde{j}\})$, where $\mathcal{Q}^1_0$ is obtained from $\mathcal{Q}_0$ as Q''_0 from Q_0.
Therefore $\mathrm{initConfig}(C'[Q_0])$ and $\mathrm{initConfig}(C'[C[[\![L]\!]]])$ satisfy the desired invariant.

- When the trace of $C'[Q_0]$ executes *new* $x[a_1,\dots,a_l] : T$ by (New) for $x \in S$ at step m, the corresponding trace of $C'[C[\llbracket L\rrbracket]]$ executes *let* $x[a_1,\dots,a_l] : T = cst$ *in* by (Let) at step m'. This yields $|I_\eta(T)|$ traces of $C'[Q_0]$, one for each value of $E_m(x[a_1,\dots,a_l])$, each with probability $p_m = p_{m-1}/|I_\eta(T)|$. In contrast, this yields a single trace of $C'[C[\llbracket L\rrbracket]]$, with probability $p'_{m'} = p'_{m'-1}$.
  Moreover, there exists no $y[a'_1,\dots,a'_{l'}] \in \mathrm{Dom}(E'_{m'})$ such that $y[a'_1,\dots,a'_{l'}] \xrightarrow{\text{var}}_{E_m} x[a_1,\dots,a_l]$. Otherwise, by the first point of the invariant, before the definition of $x[a_1,\dots,a_l]$, there would exist M_V such that $y[a'_1,\dots,a'_{l'}] \xrightarrow{\text{var}}_{E_{m-1}} M_V$ and $M_V \in \mathrm{Dom}(E_{m-1})$. Since E_m is an extension of E_{m-1}, $y[a'_1,\dots,a'_{l'}] \xrightarrow{\text{var}}_{E_m} M_V$. Since $\xrightarrow{\text{var}}_{E_m}$ is injective, $M_V = x[a_1,\dots,a_l]$. This yields a contradiction, since $M_V \in \mathrm{Dom}(E_{m-1})$ but $x[a_1,\dots,a_l] \notin \mathrm{Dom}(E_{m-1})$ by Invariant 4. ($x[a_1,\dots,a_l]$ cannot be defined several times in a trace.)
  It is then easy to see that the invariant is satisfied.
- When the trace of $C'[Q_0]$ executes $\sigma_{\mathrm{i}}P_M$ for $M \in \mathcal{M}$, the corresponding trace of $C'[C[\llbracket L\rrbracket]]$ executes $\sigma_{\mathrm{i}}(\overline{d_{M,1}}\langle\rangle; d_{M,1}(); \dots \overline{d_{M,l}}\langle\rangle; d_{M,l}(); \overline{d_M}\langle \sigma_M x_{1,M},\dots,\sigma_M x_{m,M}\rangle; d_M(y : \mathit{bitstring}); C_M[y])$ where $\sigma_{\mathrm{i}} = \{\widetilde{a}/\widetilde{i}\}$, $\widetilde{i}$ is the sequence of current replication indexes at P_M, and $BL(M) = (j_0,\dots,j_l)$.
  For $k \le l$, let a_k be such that $E_m, \mathrm{addstart}_{k,M}(\mathrm{num}_{k,M}(\sigma_{\mathrm{i}}(\mathrm{im\ index}_k(M))))) \Downarrow a_k$ and let $\widetilde{b_k}$ be such that $E_m, \sigma_{\mathrm{i}}(\mathrm{im\ index}_k(M)) \Downarrow \widetilde{b_k}$.
  Let m'_k be the step of the trace of $C'[C[\llbracket L\rrbracket]]$ after executing $\overline{\sigma_{\mathrm{i}} d_{M,k}}\langle\rangle; \sigma_{\mathrm{i}} d_{M,k}()$, where $d_{M,k} = d_{j_0,\dots,j_{k-1}}[\mathrm{convindex}(k,M)]$. We show by induction on k that for all k', $y_{(j_0,\dots,j_{k-1}),k'}[a_1,\dots,a_k] \in \mathrm{Dom}(E'_{m'_k})$ and that the invariant is satisfied at step m'_k except that $\sigma_{\mathrm{i}}(\overline{d_{M,1}}\langle\rangle; d_{M,1}(); \dots; \overline{d_{M,k}}\langle\rangle; d_{M,k}())$ has been removed from $P'_{m'_k}$. Let $z_{kk'} = \mathrm{varImL}(y_{((j_0,\dots,j_{k-1}),k'}, M)$. We have $y_{(j_0,\dots,j_{k-1}),k'}[a_1,\dots,a_k] \xrightarrow{\text{var}}_{E_m} z_{kk'}[\widetilde{b_k}]$. Moreover, $z_{kk'}[\widetilde{b_k}] \in \mathrm{Dom}(E_m)$ since $z_{kk'}[\sigma_{\mathrm{i}}(\mathrm{im\ index}_k(M))]$ occurs in $\sigma_{\mathrm{i}}M$, and $\sigma_{\mathrm{i}}M$ is successfully evaluated in the trace of $C'[Q_0]$. We distinguish two cases:
  - First case: $y_{((j_0,\dots,j_{k-1}),k'}[a_1,\dots,a_k] \in \mathrm{Dom}(E'_{m'_{k-1}})$. By the invariant at step m'_{k-1}, we have $d_{j_0,\dots,j_{k-1}}[a_1,\dots,a_k](); \overline{d_{j_0,\dots,j_{k-1}}[a_1,\dots,a_k]}\langle\rangle \in Q^2_{m'_{k-1}}$. So we can execute $\overline{\sigma_{\mathrm{i}} d_{M,k}}\langle\rangle; \sigma_{\mathrm{i}} d_{M,k}()$ by two (Output) steps, without changing the environment, so $y_{(j_0,\dots,j_{k-1}),k'}[a_1,\dots,a_k] \in \mathrm{Dom}(E'_{m'_k})$ and the invariant is satisfied at step m'_k except that $\sigma_{\mathrm{i}}(\overline{d_{M,1}}\langle\rangle; d_{M,1}(); \dots \overline{d_{M,k}}\langle\rangle; d_{M,k}())$ is removed from $P'_{m'_k}$.
  - Second case: $y_{(j_0,\dots,j_{k-1}),k'}[a_1,\dots,a_k] \notin \mathrm{Dom}(E'_{m'_{k-1}})$. By induction hypothesis, $y_{(j_0,\dots,j_{k-2}),k'}[a_1,\dots,a_{k-1}] \in \mathrm{Dom}(E'_{m'_{k-1}})$. By the invariant at step m'_{k-1}, $\mathrm{redRepl}(a_k, \mathrm{relay}(L^{a_1,\dots,a_{k-1}}_{j_0,\dots,j_{k-1}})) \in \mathcal{Q}^2_{m'_{k-1}}$ and $\mathrm{redRepl}(a_k, \llbracket L^{a_1,\dots,a_{k-1}}_{j_0,\dots,j_{k-1}}\rrbracket) \in \mathcal{Q}^3_{m'_{k-1}}$. By (Output) twice, we send an empty message on $d_{j_0,\dots,j_{k-1}}[a_1,\dots,a_k]$ and on $c_{j_0,\dots,j_{k-1}}[a_1,\dots,a_k]$. By (New), we define $y_{(j_0,\dots,j_{k-1}),k'}[a_1,\dots,a_k]$ for each k'. We choose $E_m(z_{kk'}[\widetilde{b_k}])$ as value of $y_{(j_0,\dots,j_{k-1}),k'}[a_1,\dots,a_k]$ (with probability $\frac{1}{|I_\eta(T)|}$ where T is the type of $y_{(j_0,\dots,j_{k-1}),k'}$). Finally, by (Output) twice, we send an empty message on $c_{j_0,\dots,j_{k-1}}[a_1,\dots,a_k]$ and on $d_{j_0,\dots,j_{k-1}}[a_1,\dots,a_k]$. Then the invariant is satisfied at step m'_k except that $\sigma_{\mathrm{i}}(\overline{d_{M,1}}\langle\rangle; d_{M,1}(); \dots \overline{d_{M,k}}\langle\rangle; d_{M,k}())$ is removed from $P'_{m'_k}$. (Note that the probability of the trace of $C'[C[\llbracket L\rrbracket]]$ is divided by $\prod_{k'} |I_\eta(T_{(j_0,\dots,j_{k-1}),k'})|$ where $T_{(j_0,\dots,j_{k-1}),k'}$ is the type of $y_{(j_0,\dots,j_{k-1}),k'}[a_1,\dots,a_k]$. This is what is required by the invariant since $y_{(j_0,\dots,j_{k-1}),k'}[a_1,\dots,a_k]$ is defined at step m'_k but was not at step m'_{k-1}.)

  So $y_{(j_0,\dots,j_{k-1}),k'}[a_1,\dots,a_k] \in \mathrm{Dom}(E'_{m'_l})$ for all $k \le l$ and k', and the invariant is satisfied at step m'_l except that $\sigma_{\mathrm{i}}(\overline{d_{M,1}}\langle\rangle; d_{M,1}(); \dots \overline{d_{M,l}}\langle\rangle; d_{M,l}())$ is removed from $P'_{m'_l}$. Let a be such that $E_m, \sigma_{\mathrm{i}}M \Downarrow a$. Let m'' be the step of the trace of $C'[C[\llbracket L\rrbracket]]$ after executing $\sigma_{\mathrm{i}}(\overline{d_M}\langle \sigma_M x_{1,M},\dots,\sigma_M x_{l',M}\rangle; d_M(y : \mathit{bitstring}))$ with $l' = \mathrm{nInput}_M$. By the invariant, we have two cases:
  - First case: we have $\mathrm{relay}(L^{a_1,\dots,a_l}_{j_0,\dots,j_l}) \in \mathcal{Q}^2_{m'_l}$ and $\llbracket L^{a_1,\dots,a_l}_{j_0,\dots,j_l}\rrbracket \in \mathcal{Q}^3_{m'_l}$.
    After two applications of (Output), the value of $E'_{m''}(x_{(j_0,\dots,j_l),k'}[a_1,\dots,a_l])$ is set to the value sent by the process $\sigma_{\mathrm{i}}\overline{d_M}\langle \sigma_M x_{1,M},\dots,\sigma_M x_{l',M}\rangle$, that is, $\sigma_{\mathrm{i}}\sigma_M x_{k',M}$, so we have $E_m, \sigma_{\mathrm{i}}\sigma_M x_{k',M} \Downarrow$

$E'_{m''}(x_{(j_0,\dots,j_l),k'}[a_1,\dots,a_l])$ for each k'. Since $M = \sigma_M N_M$ and $E'_{m'_l}(y_{(j_0,\dots,j_{k-1}),k'}[a_1,\dots,a_k]) = E_m(z_{kk'}[\widetilde{b_k}])$ for all $k \leq l$ and k', we have $E'_{m''}, N_M \Downarrow a$, hence $E'_{m''}(y[\widetilde{a}]) = a$.

- <u>Second case:</u> we have $d_{j_0,\dots,j_l}[a_1,\dots,a_l](x_{(j_0,\dots,j_l),1} : T_{(j_0,\dots,j_l),1},\dots,x_{(j_0,\dots,j_l),l'} : T_{(j_0,\dots,j_l),l'})$; $\overline{d_{j_0,\dots,j_l}[a_1,\dots,a_l]}\langle r\rangle \in \mathcal{Q}^2_{m'_l}$ and there exist $M' \in \mathcal{M}$ and $\widetilde{a'}$ such that $E_m, M'\{\widetilde{a'}/\widetilde{i'}\} \Downarrow r$, $E_m, \mathrm{convindex}(l, M')\{\widetilde{a'}/\widetilde{i'}\} \Downarrow a_1,\dots,a_l$, and $BL(M') = (j_0,\dots,j_l)$, where $\widetilde{i'}$ is the sequence of current replication indexes at M'.
  In this case, $\mathrm{convindex}(l, M')\{\widetilde{a'}/\widetilde{i'}\} =_{E_m} \mathrm{convindex}(l, M)\{\widetilde{a}/\widetilde{i}\}$, so, as shown in the proof that $\xrightarrow{\text{var}}_E$ is a function, $\mathrm{index}_l(M')\{\widetilde{a'}/\widetilde{i'}\} =_{E_m} \mathrm{index}_l(M)\{\widetilde{a}/\widetilde{i}\} =_{E_m} \widetilde{b_l}$ and M' and M share the first l sequences of random variables, that is, all sequences of random variables, or $m_l = 0$ and $M = M'$. Moreover, $BL(M) = BL(M') = (j_0,\dots,j_l)$, so $N_M = N_{M'}$.
  If $m_l = 0$ and $M = M'$, $\widetilde{a'} = \widetilde{a}$, so $E_m, \sigma_{\mathrm{i}} M \Downarrow r$, so $r = a$.
  Otherwise, by Hypothesis H′4.3, there exists a term M_0 such that $M = (\mathrm{index}_l(M))M_0$, $M' = (\mathrm{index}_l(M'))M_0$, and M_0 does not contain the current replication indexes at M or M'. Then $a =_{E_m} M\{\widetilde{a}/\widetilde{i}\} =_{E_m} M_0\{\widetilde{b_l}/\widetilde{i''}\} =_{E_m} M'\{\widetilde{a'}/\widetilde{i'}\} =_{E_m} r$ where $\widetilde{i''}$ is the sequence of current replication indexes at definition of $z_{lk',M}$ for any k'.

  In all cases, we obtain therefore $E'_{m''}(y[\widetilde{a}]) = a$, so $\sigma_{\mathrm{i}} C_M[y]$ in the trace of $C'[C[\![L]\!]]$ executes in the same way as $\sigma_{\mathrm{i}} C_M[M]$ in the trace of $C'[Q_0]$, which yields the desired invariant.
- The other cases are easy: both sides reduce in the same way.

Conversely, we show that all traces of $C'[C[\![L]\!]]$ correspond to a trace of $C'[Q_0]$ with the same relation as above. The proof follows a technique similar to the previous proof.

So $\prod_{z,a'_1,\dots,a'_{j'}} |I_\eta(T)|$ traces of $C'[Q_0]$, each of probability p_m, correspond to one trace of $C'[C[\![L]\!]]$ with probability $p'_{m'} = p_m \times \prod_{z,a'_1,\dots,a'_{j'}} |I_\eta(T)|$. Moreover, for all channels c and bitstrings a, $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ executes $\overline{c}\langle a\rangle$ immediately if and only if $E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ executes $\overline{c}\langle a\rangle$ immediately. So $\Pr[C'[Q_0] \rightsquigarrow_\eta \overline{c}\langle a\rangle] = \Pr[C'[C[\![L]\!]] \rightsquigarrow_\eta \overline{c}\langle a\rangle]$. Hence $Q_0 \approx_0^V C[\![L]\!]$. □

Lemma 12. $Q'_0 \approx_0^V C[\![R]\!]$

Proof sketch. The proof uses the same technique as the proof of Lemma 11. The main addition is that, in contrast to L, R may contain functional processes that are more complex than just terms. In order to handle them, we need to define a relation between variables of Q'_0 and variables of R defined by *let* or *new* in functional processes: when y is such a variable, $y[a_1,\dots,a_l] \xrightarrow{\text{var}}_E \mathrm{varImR}(y, M)[\widetilde{a'}]$ where for all $k \leq l$, $E, \mathrm{addstart}_{k,M}(\mathrm{num}_{k,M}(\mathrm{im}\ \mathrm{index}_k(M)\{\widetilde{a'}/\widetilde{i}\})) \Downarrow a_k$ and $\widetilde{i}$ is the sequence of current replication indexes at M. The relation $\xrightarrow{\text{var}}_E$ is not a function for these variables, but we can show that when $y[a_1,\dots,a_l]$ is related to several variables, these variables hold the same value at runtime.

The most delicate case is that of *find* functional processes

$$FP = \mathit{find}\ (\bigoplus_{j=1}^{m} \widetilde{u_j} \leq \widetilde{n_j}\ \mathit{suchthat}\ \mathit{defined}(z_{j1}[\widetilde{u_{j1}}],\dots,z_{jl_j}[\widetilde{u_{jl_j}}]) \wedge M_j\ \mathit{then}\ FP_j)\ \mathit{else}\ FP'$$

where for each k, $\widetilde{u_{jk}}$ is the concatenation of the prefix of the current replication indexes of length l'_0 and of a non-empty prefix of $\widetilde{u_j}$. When executing such a *find* process, $[\![R]\!]$ tests the value of $z_{jk}[a_1,\dots,a_{l'_1}]$ for all indexes of $a_1,\dots,a_{l'_1}$ such that $a_1,\dots,a_{l'_0}$ correspond to a prefix of the current replication indexes. Correspondingly, $\mathrm{transf}_{\phi,C_M}(FP)$ tests the values of all variables that are related to $z_{jk}[a_1,\dots,a_{l'_1}]$ by $\xrightarrow{\text{var}}$. □

Lemma 13. *Process Q'_0 satisfies Invariant 1.*

Proof. Process Q'_0 satisfies Invariant 1 since all newly created definitions concern fresh variables; for variables of Q'_0 that correspond to variables defined by *new* or by an input in R, there is a single definition for each of them in Q'_0; for variables of Q'_0 that correspond to variables defined by *let* in R, there are several definitions only when there are several definitions of these variables in R, and since $[\![R]\!]$ satisfies Invariant 1, these definitions are in different branches of *if* or *find* in R, so also in Q'_0. $\Box$

Lemma 14. *Process Q'_0 satisfies Invariant 2.*

Proof. The only variable accesses created in Q'_0 come from $\mathrm{transf}_{\phi_0,C_M}(FP)$. We easily show by induction on FP that the only variable accesses created by $\mathrm{transf}_{\phi,C_M}(FP)$ and not guarded by a corresponding *find* are in im ϕ. (We do not consider variable accesses in C_M, which already existed in Q_0.) So the only variable accesses created by $\mathrm{transf}_{\phi_0,C_M}(FP_M)$ and not guarded by a corresponding *find* are in im ϕ_0. Moreover, variable accesses in im ϕ_0 are of three kinds:

1. $\mathrm{varImR}(x_{j,M}, M)[i'_1, \dots, i'_{l'}]$ which are defined in P'_M, just above $\mathrm{transf}_{\phi_0,C_M}(FP_M)$.
2. $\mathrm{varImR}(y'_{jk,M}, M)[\text{im index}_j(M)]$ where
   (a) either $\mathrm{nNew}_{j,M} > 0$ and $z_{j1,M}[\text{im index}_j(M)]$ is guaranteed to be defined, since it occurs at this point in the initial process Q_0 which satisfies Invariant 2. By the addition of *defined* conditions in *find* and the fact that $z'_{jk,M} = \mathrm{varImR}(y'_{jk,M}, M)$ is defined in Q'_0 where $z_{j1,M}$ was defined in Q_0, this implies that $\mathrm{varImR}(y'_{jk,M}, M)[\text{im index}_j(M)]$ is also defined.
   (b) or $\mathrm{nNew}_{j,M} = 0$, then im $\mathrm{index}_j(M)$ is the sequence of current replication indexes at M, and $\mathrm{varImR}(y'_{jk,M}, M)[\text{im index}_j(M)]$ is defined just above P'_M.
3. $\mathrm{varImR}(z, M)[i'_1, \dots, i'_{l'}]$ where z is defined by *let* in FP_M. Since $[\![R]\!]$ satisfies Invariant 2, accesses to $z[i_1, \dots, i_l]$ in FP_M occur under the definition of $z[i_1, \dots, i_l]$ in FP_M, so accesses to $\mathrm{varImR}(z, M)[i'_1, \dots, i'_{l'}] = \phi_0(z[i_1, \dots, i_l])$ also occur under their definition in $\mathrm{transf}_{\phi_0,C_M}(FP_M)$.

Therefore, Q'_0 satisfies Invariant 2. $\Box$

Lemma 15. *Process Q'_0 satisfies Invariant 3.*

Proof. The only newly added variable definitions are *let* $\mathrm{varImR}(x_{j,M}, M) : T_{j,M} = \sigma_M x_{j,M}$ and *new* $z'_{jk,M} : T'_{jk,M}$. Each variable $\mathrm{varImR}(x_{j,M}, M)$ has at most one definition in Q'_0. For variables $z'_{jk,M}$, when several of these definitions are added for the same variable $z'_{jk,M}$, they are added in place of the definition(s) of $z_{j1,M}$, so by hypothesis H′3.1, they occur under the same replications, so they all have the same type. Therefore, the type environment for Q'_0 is well-defined.

Assume that $M \in \mathcal{M}$ and $P_M = C_M[M]$ is the smallest process containing M. Let $\mathcal{E}_L$ be the type environment at $P_M = C_M[M]$ in Q_0; let $\mathcal{E}_R$ be the type environment at P'_M in Q'_0; let $\mathcal{E}'_L$ be the type environment at N_M in L; let $\mathcal{E}'_R$ be the type environment at FP_M in R. We know that $\mathcal{E}_L \vdash P_M$, and show that $\mathcal{E}_R \vdash P'_M$. It is then easy to see that Q'_0 is well-typed knowing that Q_0 is well-typed. We note that $\mathcal{E}_R$ is an extension of $\mathcal{E}_L$ with types for variables $\mathrm{varImR}(y'_{jk,M'}, M')$, $\mathrm{varImR}(x_{j,M'}, M')$, and $\mathrm{varImR}(z, M')$ when z is defined by *let* in $FP_{M'}$, for each $M' \in \mathcal{M}$. By Hypothesis H′3.2, $\mathcal{E}_L \vdash \sigma_M x_{j,M} : T_{j,M}$, so $\mathcal{E}_R \vdash \sigma_M x_{j,M} : T_{j,M}$, since $\mathcal{E}_R$ is an extension of $\mathcal{E}_L$. Then, in order to show $\mathcal{E}_R \vdash P'_M$, it is enough to show $\mathcal{E}_R \vdash \mathrm{transf}_{\phi_0,C_M}(FP_M)$.

We say that ϕ is well-typed when $z[\widetilde{M}] \in \mathrm{Dom}(\phi)$ and $\mathcal{E}'_R \vdash z[\widetilde{M}] : T'$ implies $\mathcal{E}_R \vdash \phi(z[\widetilde{M}]) : T'$.

First, it is easy to show by induction on M' that for all well-typed ϕ, for all M' such that $\mathcal{E}'_R \vdash M' : T$, we have $\mathcal{E}_R \vdash \phi(M') : T$.

Next, we show that for all well-typed ϕ, if $\mathcal{E}'_R \vdash [\![FP']\!]^{\widetilde{j}}_{\widetilde{i}}$ and the type of the result of FP' is the type of N_M, then $\mathcal{E}_R \vdash \mathrm{transf}_{\phi,C_M}(FP')$, by induction on FP'.

- If $FP' = M'$, we have to show that $\mathcal{E}_R \vdash C_M[\phi(M')]$. Let T such that $\mathcal{E}_L \vdash M : T$.
  We have $M = \sigma_M N_M$, so if N_M contains a function symbol, $\mathcal{E}'_L \vdash N_M : T$. If $N_M = x_{j,M}$, $M = \sigma_M x_{j,M}$ is of type $T_{j,M}$ by Hypothesis H′3.2, so $T = T_{j,M}$, hence we also have $\mathcal{E}'_L \vdash N_M : T$.

If $N_M = y_{jk,M}$, $M = \sigma_M y_{jk,M} = z_{jk,M}[\text{im index}_j(M)]$ is of type $T_{jk,M}$ by Hypothesis H′3.1, so $T = T_{jk,M}$ and we also have $\mathcal{E}'_L \vdash N_M : T$.
By hypothesis, we have then $\mathcal{E}'_R \vdash M' : T$, so $\mathcal{E}_R \vdash \phi(M') : T$. Since $\mathcal{E}_L \vdash C_M[M]$ with $\mathcal{E}_L \vdash M : T$, by a substitution lemma, we conclude that $\mathcal{E}_R \vdash C_M[\phi(M')]$.

– The inductive cases follow easily using $\mathcal{E}'_R \vdash [\![FP']\!]^{\widetilde{j}}_{\widetilde{i}}$ and the property proved above to type terms.
In the case of *find*, we extend ϕ into ϕ' as follows. Let $\widetilde{i'}$ be the sequence of current replication indexes at M' and $\widetilde{u'}$ be a sequence formed with a fresh variable for each variable in $\widetilde{i'}$.
  • If $z_k = y'_{jk',M'}$ for some k', $\phi'(z_k[M_{k1},\dots,M_{kl'_k}]) = \text{varImR}(z_k, M')[\text{im index}_j(M')\{\widetilde{u'}/\widetilde{i'}\}]$. Since $\text{varImR}(z_k, M')$ is defined where $z_{j1,M'}$ is defined, the indexes of $\text{varImR}(z_k, M')$ are the indexes of $z_{j1,M'}$, so $\text{im index}_j(M')$ is of the suitable type. Moreover, $\widetilde{u'}$ and $\widetilde{i'}$ have the same types, so by a substitution lemma, $\text{im index}_j(M')\{\widetilde{u'}/\widetilde{i'}\}$ is of the suitable type. Moreover z_k in R and $\text{varImR}(z_k, M')$ in Q'_0 are both declared of type $T'_{jk',M'}$, so $\mathcal{E}'_R \vdash z_k[M_{k1},\dots,M_{kl'_k}] : T'_{jk',M'}$ and $\mathcal{E}_R \vdash \text{varImR}(z_k, M')[\text{im index}_j(M')\{\widetilde{u'}/\widetilde{i'}\}] : T'_{jk',M'}$.
  • If z_k is defined by *let* or by a function input, $\phi'(z_k[M_{k1},\dots,M_{kl'_k}]) = \text{varImR}(z_k, M')[\widetilde{u'}]$. $\text{varImR}(z_k, M')$ is declared under the same replications as M', so $\widetilde{u'}$ is of the suitable type. The variables z_k in R and $\text{varImR}(z_k, M')$ in Q'_0 are declared of the same type, so if $\mathcal{E}'_R \vdash z_k[M_{k1},\dots,M_{kl'_k}] : T'$ then $\mathcal{E}_R \vdash \text{varImR}(z_k, M')[\widetilde{u'}] : T'$.

So ϕ' is well-typed.
Moreover, we show that $\mathcal{E}_R \vdash \text{im index}_{j_1}(M')\{\widetilde{u'}/\widetilde{i'}\} = \text{im index}_{j_1}(M) : \mathit{bool}$. We have $z_{j_1k,M} = z_{j_1k,M'}$ since M and M' share the j_1 first sequences of random variables, so $\text{im index}_{j_1}(M')$ and $\text{im index}_{j_1}(M)$ are of the same type, since they are both used as indexes of $z_{j_1k,M}$. Since $\widetilde{u'}$ and $\widetilde{i'}$ are of the same type, by a substitution lemma, $\text{im index}_{j_1}(M')\{\widetilde{u'}/\widetilde{i'}\}$ and $\text{im index}_{j_1}(M)$ are of the same type, which yields the desired result.

It is easy to see that ϕ_0 is well-typed. Moreover $\mathcal{E}'_R \vdash [\![FP_M]\!]^{\widetilde{j}}_{\widetilde{i}}$ and the type of the result of FP_M is the type of N_M by Hypothesis H0, so $\mathcal{E}_R \vdash \text{transf}_{\phi_0,C_M}(FP_M)$. □

Proof of Proposition 4 Invariants 1, 2, and 3 have been proved in Lemmas 13, 14, and 15 respectively. Finally, we show that $Q_0 \approx^V Q'_0$. After renaming variables so that V and C do not contain variables of L and R, by Lemmas 1, 11, and 12, $Q_0 \approx^V_0 C[[\![L]\!]] \approx^V C[[\![R]\!]] \approx^V_0 Q'_0$, so by transitivity $Q_0 \approx^V Q'_0$. □

B.6 Proofs for Section 4

Proof of Proposition 5 Let C be an acceptable context for $Q \mid Q_x$, $Q \mid Q'_x$, $\emptyset$. We relate the traces of $C[Q \mid Q_x]$ and $C[Q \mid Q'_x]$ as follows:

– If a trace of $C[Q \mid Q_x]$ never executes $\overline{c}\langle x[i_1,\dots,i_m]\rangle$, then we obtain a trace of $C[Q \mid Q'_x]$ with the same probability, by just replacing Q_x with Q'_x and subprocesses of Q_x with the corresponding subprocess of Q'_x.
– Otherwise, the considered trace of $C[Q \mid Q_x]$ executes $\overline{c}\langle x[i_1,\dots,i_m]\rangle$ exactly once, with $E(i_1) = a_1,\dots,E(i_m) = a_m$, and $E(x[a_1,\dots,a_m]) = a$, where E is the environment when $\overline{c}\langle x[i_1,\dots,i_m]\rangle$ is executed. By hypothesis, the definition of $x[a_1,\dots,a_m]$ in this trace is either a restriction *new* $x[a_1,\dots,a_m] : T$, or an assignment *let* $x[a_1,\dots,a_m] : T = z[M_1,\dots,M_l]$ with $E, M_k \Downarrow a'_k$ for all $k \leq l$, and the definition of $z[a'_1,\dots,a'_l]$ in this trace is *new* $z[a'_1,\dots,a'_l] : T$.
We build $|I_\eta(T)|$ traces of $C[Q \mid Q'_x]$ from this trace, by choosing any value of $I_\eta(T)$ for the restriction *new* $x[a_1,\dots,a_m] : T$ or *new* $z[a'_1,\dots,a'_l] : T$ defined above, and the value a for the restriction *new* $y : T$ of Q'_x. By definition of S, these traces are the same as the trace of $C[Q \mid Q_x]$ except

perhaps for values of variables in S, and for the process Q'_x instead of Q_x. The probability of each of these traces is $1/|I_\eta(T)|$ times the probability of the considered trace of $C[Q \mid Q_x]$, since these traces choose one more random number in $I_\eta(T)$ than the trace of $C[Q \mid Q_x]$.

Moreover, all traces of $C[Q \mid Q'_x]$ are obtained by the previous construction. (To show that, we rebuild a trace of $C[Q \mid Q_x]$ from the trace of $C[Q \mid Q'_x]$ by the reverse construction of the one detailed above.)

For each configuration $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ of the trace of $C[Q \mid Q_x]$, and corresponding configuration $E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ of the trace of $C[Q \mid Q'_x]$, for all channels c and bitstrings a, $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ executes $\overline{c}\langle a\rangle$ immediately if and only if $E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ executes $\overline{c}\langle a\rangle$ immediately.

Therefore $\Pr[C[Q \mid Q_x] \rightsquigarrow_\eta \overline{c}\langle a\rangle] = \Pr[C[Q \mid Q'_x] \rightsquigarrow_\eta \overline{c}\langle a\rangle]$, so $Q \mid Q_x \approx_0 Q \mid Q'_x$. □

Proof of Proposition 6 Let C be an acceptable context for $Q \mid Q_x$, $Q \mid Q'_x$, $\emptyset$.

We first exclude traces $\mathcal{T}$ such that $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) = \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a'}])$ and $\widetilde{a} \neq \widetilde{a'}$. These traces have negligible probability by hypothesis, since $C[\_ \mid Q_x]$ is an acceptable context for Q, 0, $\{x\}$. So this removal does not change the result.

For the remaining traces, when $\widetilde{a} \neq \widetilde{a'}$, $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) \neq \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a'}])$, so the definitions of $x[\widetilde{a}]$ and $x[\widetilde{a'}]$ do not come from a single execution of the same restriction. (So $x[\widetilde{a}]$ and $x[\widetilde{a'}]$ are independent random numbers.) Then we can apply a proof similar to that of Proposition 5, except that we replace each tested value of $x[\widetilde{a'}]$ with independent random numbers instead of single one. □

Proof of Lemma 2 Let us prove the result for one-session secrecy. (The proof is essentially the same for secrecy.) $[\,] \mid Q_x$ and $[\,] \mid Q'_x$ are acceptable contexts for Q, Q', $\{x\}$ (after renaming $i, i_1, \dots, i_m$ so that they do not occur in Q and Q'). So by Lemma 1, $Q \mid Q_x \approx^{\{x\}} Q' \mid Q_x$ and $Q \mid Q'_x \approx^{\{x\}} Q' \mid Q'_x$. A fortiori, $Q \mid Q_x \approx Q' \mid Q_x$ and $Q \mid Q'_x \approx Q' \mid Q'_x$. So by transitivity of $\approx$, $Q' \mid Q_x \approx Q' \mid Q'_x$. □

C Optimizations for $\mathbf{transf}_{\phi,C_M}(FB)$

We can apply two optimizations to the definition of $\mathrm{transf}_{\phi,C_M}(FB)$:

- When $\mathrm{im\ index}_{j_1}(M')$ is a prefix of $\widetilde{i'}$, $\mathrm{im\ index}_{j_1}(M')\{\widetilde{u'}/\widetilde{i'}\}$ is a prefix of $\widetilde{u'}$, so the equality $\mathrm{im\ index}_{j_1}(M')\{\widetilde{u'}/\widetilde{i'}\} = \mathrm{im\ index}_{j_1}(M)$ defines the value of a prefix of $\widetilde{u'}$. We simply substitute the fixed elements of $\widetilde{u'}$ with their value, and remove them from the sequence of variables to be looked up by *find*.
- When all variables z_k are $y_{jk',M'}$ for some j, k', and M', with $\max j = j_0$, we use the following definition instead:

$$\begin{aligned}&\mathrm{transf}_{\phi,C_M}(\widetilde{i} \leq \widetilde{n}\ \textit{suchthat defined}(z_k[M_{k1},\dots,M_{kl'_k}]_{1\leq k\leq l}) \wedge M_1\ \textit{then}\ FP') =\\ &\bigoplus\nolimits_{M'\in\mathcal{M}'} \widetilde{u'} \leq \widetilde{n'}\ \textit{suchthat defined}(\phi'(z_k[M_{k1},\dots,M_{kl'_k}])_{1\leq k\leq l}) \wedge\\ &\qquad \mathrm{im}\ (\rho_{j_0-1}(M')\circ\ldots\circ\rho_{j_1}(M'))\{\widetilde{u'}/\widetilde{i'}\} = \mathrm{im\ index}_{j_1}(M) \wedge \phi'(M_1)\ \textit{then}\ \mathrm{transf}_{\phi',C_M}(FP')\end{aligned}$$

  where j_1 is the length of the prefix of the current replication indexes that occurs in $M_{k1},\dots,M_{kl'_k}$ (by hypothesis H7); $\mathcal{M}'$ is the set of $M' \in \mathcal{M}$ such that $\mathrm{varImR}(z_k, M')$ is defined for $k \leq l$ and M' and M share the j_1 first sequences of random variables; $\widetilde{i'}$ is the sequence of current replication indexes at the definition of $z_{j_0k,M'}$; $\widetilde{u'}$ is a sequence formed with a fresh variable for each variable in $\widetilde{i'}$; $\widetilde{n'}$ is the sequence of bounds of replications above the definition of $z_{j_0k,M'}$; ϕ' is an extension of ϕ with $\phi'(z_k[M_{k1},\dots,M_{kl'_k}]) = \mathrm{varImR}(z_k, M')[\mathrm{im}\ (\rho_{j_0-1}(M')\circ\ldots\circ\rho_j(M'))\{\widetilde{u'}/\widetilde{i'}\}]$ if $z_k = y'_{jk,M'}$. The composition $\rho_{j_0-1}(M')\circ\ldots\circ\rho_j(M')$ computes the indexes of $z'_{jk',M'}$ for any k' from the indexes of $z'_{j_0k'',M'}$ for any k''.

When several terms $M' \in \mathcal{M}$ share the first j_0 sequences of random variables, they generate the same ϕ', so only one *find* branch needs to be added for all of them, which can reduce considerably the number of *find* branches to add.
An optimization similar to the first one above also applies to this case, when im $(\rho_{j_0-1}(M') \circ \ldots \circ \rho_{j_1}(M'))$ is a prefix of $\widetilde{i'}$.