Computationally Sound Mechanized Proofs of Correspondence Assertions

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

Abstract

We present a new mechanized prover for showing correspondence assertions for cryptographic protocols in the computational model. Correspondence assertions are useful in particular for establishing authentication. Our technique produces proofs by sequences of games, as standard in cryptography. These proofs are valid for a number of sessions polynomial in the security parameter, in the presence of an active adversary. Our technique can handle a wide variety of cryptographic primitives, including shared- and public-key encryption, signatures, message authentication codes, and hash functions. It has been implemented in the tool CryptoVerif and successfully tested on examples from the literature.

1. Introduction

Correspondence assertions on cryptographic protocols are properties of the form "if some events have been executed, then some other events have been executed", where each event corresponds to a certain point in the protocol, possibly with arguments. An event can be formalized by a special instruction event $e(M_1, \dots, M_m)$, which simply records that the event $e(M_1, \dots, M_m)$ has been executed. Woo and Lam [63] introduced correspondence assertions to express the authentication properties of cryptographic protocols, such as "if B terminates a run of the protocol, apparently with A, then A has started a run of the protocol, apparently with B." This property can be written more formally "if event $Bterminates(A)$ has been executed, then event $Astarts(B)$ has been executed", where event $Bterminates(X)$ occurs at the point where B terminates a run and he thinks he talks to X, and event $Astarts(Y)$ occurs at the point where A starts a run with Y. Correspondence assertions have become a standard tool for reasoning on cryptographic protocols.

The main novelty of our work lies in the model in which we prove correspondence assertions. Indeed, there are two main models for cryptographic protocols. In the computational model, cryptographic primitives are functions on bitstrings and the adversary is a polynomial-time probabilistic Turing machine. In this realistic model, proofs are usually manual. In the formal, Dolev-Yao model, cryptographic primitives are considered as perfect blackboxes represented by function symbols, and the adversary is restricted to compute with these blackboxes. There already exist several techniques for proving correspondence assertions automatically in this abstract model, e.g. [18, 36]. However, in general, these proofs are not sound with respect to the computational model.

Since the seminal paper by Abadi and Rogaway [6], there has been much interest in relating both models [4, 11, 14, 30, 31, 38, 39, 50, 51], 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, for symmetric encryption, key cycles have to be excluded, or a specific security definition of encryption is needed [8].)

In this paper, we adopt a different approach: our tool proves correspondences directly in the computational model. In order to achieve such proofs, we extend our previous approach for secrecy [20, 21]. We produce proofs by sequences of games, as used by cryptographers [17, 57–59]: the initial game represents the protocol, for which we want to prove that the probability of breaking a certain correspondence is negligible; 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 can directly be shown to be negligible from the form of the game. The desired probability is then negligible in the initial game.

In order to extend our approach to correspondence assertions, we slightly extend the calculus that we use to represent games, so that it can specify events. The game transformations that we used for secrecy can also be used for correspondences, without change. However, we still need to check that the correspondence holds on the final game. So, we introduce a rich language of correspondence assertions, and show how to check them automatically. This language allows one to specify both injective correspondences (if some event has been executed n times, then some other events have been executed at least n times) and non-injective correspondences (if some events have been executed, then some other events have been executed at least once), as well as properties of the form "if some events have been executed, then some formula holds".

Moreover, we also show how to use correspondences in order to prove mutual authentication and authenticated key exchange. Mutual authentication is an immediate consequence of correspondences. The situation is more subtle for authenticated key exchange: intuitively, we need to prove the secrecy of the key. Since the key is shared between two participants of the protocol, the secrecy of the key is not simply the secrecy of a single variable, as we could prove in [20, 21]. However, we show that by combining correspondences with the secrecy of the variable that contains the key for one of the participants of the protocol, we can prove the standard notion of authenticated key exchange.

The prover succeeds in a fully automatic way for many examples. For delicate cases, our prover allows the user to indicate the main game transformations to perform, such as applying the security of a certain cryptographic primitive for a certain secret key. Importantly, the prover is always sound, whatever indications the user gives.

The verification of correspondences has been implemented in our prover CryptoVerif (19200 lines of Ocaml for version 1.06 of CryptoVerif), 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. [31, 39, 51], make it possible to use Dolev-Yao provers in order to prove correspondences in the computational model. In particular, a tool [29] has been built based on [31] in order to make computational proofs using the Dolev-Yao prover AVISPA, for protocols that use public-key encryption and signatures. However, computational soundness 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 [10–12] 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. This framework shares some limitations with the computational soundness results, for instance the exclusion of key cycles and the fact that symmetric encryption has to be authenticated. It relates the computational model to a non-standard version of the Dolev-Yao model, in which the length of messages is still present. It has been used for a computationally-sound machine-checked proof of the Needham-Schroeder-Lowe protocol [60].

Canetti and Herzog [26] show how a Dolev-Yao-style symbolic analysis can be used to prove security properties of protocols (including authentication) within the framework of universal composability [24], for a restricted class of protocols using public-key encryption as only cryptographic primitive. Then, they use the automatic Dolev-Yao verification tool ProVerif [19] for verifying protocols in this framework.

Canetti et al. [25] use the framework of time-bounded task-PIOAs (Probabilistic Input/Output Automata) for proving cryptographic protocols in the computational model. This framework allows them to combine probabilistic and non-deterministic behaviors.

Lincoln et al. [46, 47, 49, 52, 56] developed a probabilistic polynomial-time calculus for the analysis of security protocols. This calculus comes with a notion of process equivalence, used in particular to prove authentication properties in [47]. This calculus resembles ours in that both are probabilistic polynomial-time variants of the pi calculus. (The restriction chooses a fresh random number. The replication is polynomially bounded.) However, it differs from our calculus since it uses an explicit probabilistic scheduler while, in our calculus, the adversary schedules the processes. Our calculus also adds arrays in order to store all values of variables, which is key to our proofs, as we shall see in the following of the paper.

Datta et al. [32, 33] have designed a computationally sound logic that enables them to prove computational security properties using a logical deduction system.

Corin and Hartog [28] use a probabilistic Hoare-style logic for formalizing game-based cryptographic proofs.

All these frameworks can be used to prove security properties of protocols in the computational sense, but except for [26] which relies on a Dolev-Yao prover and for the machine-checked proofs of [60], they have not been mechanized up to now, as far as we know.

Other works provide proofs in the computational model, but only for secrecy. Laud [43] designed an automatic analysis for protocols using shared-key encryption, with passive adversaries. He extended it to active adversaries, but with only one session of the protocol [44]. The type system of [9, 45] handles shared-key and public-key encryption, with an unbounded number of sessions. This system relies on the Backes-Pfitzmann-Waidner library.

Barthe, Cerderquist, and Tarento [13, 61] 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, to build a detailed enough proof for the theorem prover to check it.

Halevi [37] 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 recalls the process calculus that we use to represent games and extends it with events. Section 3 defines the correspondence assertions that we prove. Section 4 recalls the definition of observational equivalence and extends it with events. Section 5 illustrates on an example the game transformations used in our proofs. Section 6 details how we prove correspondences. Section 7 shows how to prove standard notions of authentication and authenticated key exchange using correspondences. Finally, Section 8 summarizes our experimental results and Section 9 concludes. The appendix contains details on the semantics of the calculus, the proof engine we use for reasoning on games, the proofs of our results, and our experiments.

2. A Calculus for Games

In this section, we review the process calculus defined in [20, 21] in order to model games used in computational security proofs. This calculus has been carefully designed to make the automatic proof of cryptographic protocols easier. We extend this calculus with parametric events, which serve in the definition of correspondences.

We illustrate this calculus on the following example, inspired by the corrected Woo-Lam public key protocol [64]:

$$B \to A : (N, B)$$
$$A \to B : \{pk_A, B, N\}_{sk_A}$$

This protocol is a simple nonce challenge: B sends to A a fresh nonce N and its identity. A replies by signing the nonce N, B's identity, and A's public key (which we use here instead of A's identity for simplicity: this avoids having to relate identities and keys; the prover can obviously also handle the version with A's identity). The signatures are assumed to be (existentially) unforgeable under chosen message attacks (UF-CMA) [35], so, when B receives the signature, B is convinced that A is present. The signature cannot be a replay because the nonce N is signed.

In our calculus, this protocol is encoded by the following process G_0, explained below:

$$G_0 = c_0(); \mathsf{new}\ rk_A : \mathit{keyseed}; \mathsf{let}\ pk_A = \mathrm{pkgen}(rk_A)\ \mathsf{in}$$
$$\mathsf{let}\ sk_A = \mathrm{skgen}(rk_A)\ \mathsf{in}\ \overline{c_1}\langle pk_A\rangle; (Q_A \mid Q_B)$$

$$Q_A = !^{i_A \le n} c_2[i_A](x_N : \mathit{nonce}, x_B : \mathit{host});$$
$$\mathsf{event}\ e_A(pk_A, x_B, x_N); \mathsf{new}\ r : \mathit{seed};$$
$$\overline{c_3[i_A]}\langle \mathrm{sign}(\mathrm{concat}(pk_A, x_B, x_N), sk_A, r)\rangle$$
$$Q_B = !^{i_B \le n} c_4[i_B](x_{pk_A} : \mathit{pkey}); \mathsf{new}\ N : \mathit{nonce};$$
$$\overline{c_5[i_B]}\langle N, B\rangle; c_6[i_B](s : \mathit{signature});$$
$$\mathsf{if}\ \mathrm{verify}(\mathrm{concat}(x_{pk_A}, B, N), x_{pk_A}, s)\ \mathsf{then}$$
$$\mathsf{if}\ x_{pk_A} = pk_A\ \mathsf{then\ event}\ e_B(x_{pk_A}, B, N)$$

The process G_0 is assumed to run in interaction with an adversary, which also models the network. G_0 first receives an empty message on channel c_0, sent by the adversary. Then, it chooses randomly with uniform probability a bitstring rk_A in the type $\mathit{keyseed}$, by the construct $\mathsf{new}\ rk_A : \mathit{keyseed}$. A type T, such as $\mathit{keyseed}$, aims at denoting a set of bitstrings. However, the considered set of bitstrings depends on the security parameter η, which determines the length of keys. So, more precisely, a type T corresponds for each value of η to a set of bitstrings denoted by $I_\eta(T)$. Then, G_0 generates the public key pk_A corresponding to the coins rk_A, by calling the public-key generation algorithm pkgen. Similarly, G_0 generates the secret key sk_A by calling skgen. It outputs the public key pk_A on channel c_1, so that the adversary has this public key.

After outputting this message, the control passes to the receiving process, which is part of the adversary. Several processes are then made available, which represent the roles of A and B in the protocol: the process $Q_A \mid Q_B$ is the parallel composition of Q_A and Q_B; it makes simultaneously available the processes defined in Q_A and Q_B. Let Q'_A and Q'_B be such that $Q_A = !^{i_A \le n} Q'_A$ and $Q_B = !^{i_B \le n} Q'_B$. The replication $!^{i_A \le n} Q'_A$ represents n copies of the process Q'_A, indexed by the replication index i_A. (The symbol n corresponds to an integer $I_\eta(n)$ for each value of the security parameter η; $I_\eta(n)$ is required to be a polynomially bounded function of η.) The process Q'_A begins with an input on channel $c_2[i_A]$; the channel is indexed with i_A so that the adversary can choose which copy of the process Q'_A receives the message by sending it on channel $c_2[i_A]$ for the appropriate value of i_A. The situation is similar for Q'_B, which expects a message on channel $c_4[i_B]$. The adversary can then run each copy of Q'_A or Q'_B simply by sending a message on the appropriate channel $c_2[i_A]$ or $c_4[i_B]$.

The process Q'_B first expects on channel $c_4[i_B]$ a message x_{pk_A} in the type pkey of public keys. This message is not really part of the protocol. It serves for starting a new session of the protocol, in which B interacts with the participant of public key x_{pk_A}. For starting a session between A and B, this message should be pk_A. Then, Q'_B chooses randomly with uniform probability a nonce N in the type nonce. The type nonce is *large*: a type T is *large* when the inverse of its cardinal $\frac{1}{|I_\eta(T)|}$ is negligible, so that collisions between independent random numbers chosen uniformly in a large type have negligible probability. (The probability $f(\eta)$ is *negligible* when for all polynomials q, there exists $\eta_o \in \mathbb{N}$ such that for all $\eta > \eta_0$, $f(\eta) \leq \frac{1}{q(\eta)}$. The probability $f(\eta)$ is *overwhelming* when $1 - f(\eta)$ is negligible.) Q'_B sends the message (N, B) on channel $c_5[i_B]$. The control then passes to the receiving process, included in the adversary. This process is expected to forward this message (N, B) on channel $c_2[i_A]$, but may proceed differently in order to mount an attack against the protocol.

Upon receiving a message (x_N, x_B) on channel $c_2[i_A]$, where the bitstring x_N is in the type $nonce$ and x_B in the type $host$, the process Q'_A executes the event $e_A(pk_A, x_B, x_N)$. This event does not change the state of the system. Events just record that a certain program point has been reached, with certain values of the arguments of the event. Then, Q'_A chooses randomly with uniform probability a bitstring r in the type $seed$; this random bitstring is next used as coins for the signature algorithm. Finally, Q'_A outputs the signed message $\{pk_A, x_B, x_N\}_{sk_A}$. (The function concat concatenates its arguments, with information on the length of these arguments, so that the arguments can be recovered from the concatenation.) The control then passes to the receiving process, which should forward this message on channel $c_6[i_B]$ if it wishes to run the protocol correctly.

Upon receiving a message s on $c_6[i_B]$, Q'_B verifies that the signature s is correct and, if $x_{pk_A} = pk_A$, that is, if B runs a session with A, it executes the event $e_B(x_{pk_A}, B, N)$. Our goal is to prove that, if event e_B is executed, then event e_A has also been executed. However, when B runs a session with a participant other than A, it is perfectly correct that B terminates without event e_A being executed; that is why event e_B is executed only when B runs a session with A.

In our calculus, all variables defined under a replication are implicitly arrays. For example, the variable x_N defined under $!^{i_A \leq n}$ is implicitly an array indexed by the replication index i_A: x_N is an abbreviation for $x_N[i_A]$. Similarly, x_B is an abbreviation for $x_B[i_A]$, r for $r[i_A]$, x_{pk_A} for $x_{pk_A}[i_B]$, N for $N[i_B]$, and s for $s[i_B]$. Using arrays allows us to remember the values of the variables in each copy of the processes, so that the whole state of the system is available. In our calculus, arrays replace lists often used by cryptographers in their proofs. For example, during the proof, all messages signed under sk_A would be stored in a list, and by the unforgeability of signatures, when the verification of the signature of a message succeeds, we would be sure that this message occurs in the list. In our calculus, we will store messages in arrays instead. Arrays come with a lookup construct: $\mathsf{find}\ u_1 \leq n_1, \dots, u_m \leq n_m\ \mathsf{suchthat}\ \mathsf{defined}(M_1, \dots, M_l) \wedge M\ \mathsf{then}\ P\ \mathsf{else}\ P'$ looks for indices $u_1, \dots, u_m$ such that $M_1, \dots, M_l$ are defined and M is true. When such indices are found, it executes P; otherwise, it executes P'. When several values of indices are possible, each possible value is chosen with the same probability. For example, $\mathsf{find}\ u \leq n\ \mathsf{suchthat}\ \mathsf{defined}(x_N[u]) \wedge x_N[u] = N\ \mathsf{then}\ P$ looks for an index u such that $x_N[u]$ is defined and equal to N. Here, the find construct does not occur in the initial game, but will be introduced by game transformations.

As detailed in [20, 21], we require some *well-formedness invariants* to guarantee that bitstrings are of their expected type and that arrays are used properly (that each cell of an array is assigned at most once during execution and that variables are accessed only after being initialized).

All processes of our calculus run in probabilistic polynomial time. The semantics of the calculus is defined by a probabilistic reduction relation on semantic configurations $\mathbb{C}$. We denote by $\mathrm{initConfig}(Q)$ the initial configuration associated to process Q. We refer the reader to Appendix A and [20] for additional details on this calculus and its semantics. Given a mapping ρ from variable names to bitstrings, we write $\rho, M \Downarrow a$ when the term M (built from function symbols and variables, without array accesses) evaluates to bitstring a. We denote by $\Pr[Q \rightsquigarrow \overline{c}\langle a \rangle]$ the probability that Q outputs the bitstring a on channel c after some reductions. We denote by $\mathcal{E}$ a sequence of events of the form $e(a_1, \dots, a_n)$, where e is an event symbol and $a_1, \dots, a_n$ are bitstrings. We denote by $\Pr[\exists (\mathbb{C}, \mathcal{E}), \mathrm{initConfig}(Q) \xrightarrow{\mathcal{E}} \mathbb{C} \wedge \phi(\mathbb{C}, \mathcal{E})]$ the probability that there exists a sequence of events $\mathcal{E}$ and a semantic configuration $\mathbb{C}$ such that Q reduces to $\mathbb{C}$, executing events $\mathcal{E}$ on the trace, and the logical formula $\phi(\mathbb{C}, \mathcal{E})$ holds. We denote by $\Pr[Q \rightsquigarrow \mathcal{E}] = \Pr[\exists \mathbb{C}, \mathrm{initConfig}(Q) \xrightarrow{\mathcal{E}} \mathbb{C} \wedge \mathbb{C}$ does not reduce$]$ the probability that the process Q executes exactly the sequence of events $\mathcal{E}$, in the order of $\mathcal{E}$. These probabilities depend on the security parameter η; we omit it to lighten notations.

We use an *evaluation context* C to represent the adversary. An evaluation context is a process with a hole, of one of the following forms: a hole $[\,]$, a process in parallel with an evaluation context $Q \mid C$, or a restriction $\mathsf{newChannel}\ c; C$, which limits the scope of the channel c to the context C. We denote by $C[Q]$ the process obtained by replacing the hole of C with Q. When V is a set of variables defined in Q, an evaluation context C is said to be *acceptable* for (Q, V) if and only if C does not contain events, the common variables of C and Q are in V, and $C[Q]$ satisfies the well-formedness invariants. The set V contains the variables the context is allowed to access (using find).

When P is under replications $!^{i_1 \leq n_1} \dots !^{i_m \leq n_m}$, we say that the *replication indices at* P are $i_1, \dots, i_m$. We denote by $\widetilde{i}$ a sequence of replication indices $i_1, \dots, i_m$, and by $\widetilde{M}$ a sequence of terms $M_1, \dots, M_m$. We denote by $\mathrm{fc}(P)$ the set of free channels of P, and by $\mathrm{var}(P)$ the set of variables that occur in P. We also use the notation $\mathrm{var}(\cdot)$ for contexts, terms, and formulas.

3. Definition of Correspondences

In this section, we define non-injective and injective correspondences.

3.1. Non-injective Correspondences

A non-injective correspondence is a property of the form "if some events have been executed, then some other events have been executed at least once". Here, we generalize these correspondences to implications between logical formulae $\psi \Rightarrow \phi$, which may contain events. We use the following logical formulae:

$\phi ::=$	formula
M	term
$\mathsf{event}(e(M_1, \ldots, M_m))$	event
$\phi_1 \wedge \phi_2$	conjunction
$\phi_1 \vee \phi_2$	disjunction

Terms $M, M_1, \ldots, M_m$ in formulae must not contain array accesses, and their variables are assumed to be distinct from variables of processes. The formula M holds when M evaluates to true. The formula $\mathsf{event}(e(M_1, \ldots, M_n))$ holds when the event $e(M_1, \ldots, M_n)$ has been executed. The conjunction and disjunction are defined as usual. More formally, we write $\rho, \mathcal{E} \vdash \phi$ when the sequence of events $\mathcal{E}$ satisfies the formula ϕ, in the environment ρ that maps variables to bitstrings. We define $\rho, \mathcal{E} \vdash \phi$ as follows:

$\rho, \mathcal{E} \vdash M$ if and only if $\rho, M \Downarrow \mathrm{true}$
$\rho, \mathcal{E} \vdash \mathsf{event}(e(M_1, \ldots, M_m))$ if and only if
for all $j \leq m$, $\rho, M_j \Downarrow a_j$ and $e(a_1, \ldots, a_m) \in \mathcal{E}$
$\rho, \mathcal{E} \vdash \phi_1 \wedge \phi_2$ if and only if $\rho, \mathcal{E} \vdash \phi_1$ and $\rho, \mathcal{E} \vdash \phi_2$
$\rho, \mathcal{E} \vdash \phi_1 \vee \phi_2$ if and only if $\rho, \mathcal{E} \vdash \phi_1$ or $\rho, \mathcal{E} \vdash \phi_2$

Formulae denoted by ψ are conjunctions of events.

Definition 1 The sequence of events $\mathcal{E}$ *satisfies the correspondence* $\psi \Rightarrow \phi$, written $\mathcal{E} \vdash \psi \Rightarrow \phi$, if and only if for all ρ defined on $\mathrm{var}(\psi)$ such that $\rho, \mathcal{E} \vdash \psi$, there exists an extension ρ' of ρ to $\mathrm{var}(\phi)$ such that $\rho', \mathcal{E} \vdash \phi$.

Intuitively, a sequence of events $\mathcal{E}$ satisfies $\psi \Rightarrow \phi$ when, if $\mathcal{E}$ satisfies ψ, then $\mathcal{E}$ satisfies ϕ. The variables of ψ are universally quantified; those of ϕ that do not occur in ψ are existentially quantified.

Definition 2 The process Q *satisfies the correspondence* $\psi \Rightarrow \phi$ with public variables V if and only if for all evaluation contexts C acceptable for (Q, V), $\Pr[\exists(\mathbb{C}, \mathcal{E}), \mathrm{initConfig}(C[Q]) \xrightarrow{\mathcal{E}} \mathbb{C} \wedge \mathcal{E} \not\vdash \psi \Rightarrow \phi]$ is negligible.

A process satisfies $\psi \Rightarrow \phi$ when the probability that it generates a sequence of events $\mathcal{E}$ that does not satisfy $\psi \Rightarrow \phi$ is negligible, in the presence of an adversary represented by the context C.

Example 1 Referring to the example G_0 of Section 2, the correspondence

$$\mathsf{event}(e_B(x, y, z)) \Rightarrow \mathsf{event}(e_A(x, y, z)) \tag{1}$$

means that, with overwhelming probability, for all x, y, z, if $e_B(x, y, z)$ has been executed, then $e_A(x, y, z)$ has been executed.

The correspondence

$$\mathsf{event}(e_1(x)) \wedge \mathsf{event}(e_2(x)) \Rightarrow \mathsf{event}(e_3(x)) \vee (\mathsf{event}(e_4(x, y)) \wedge \mathsf{event}(e_5(y, z)))$$

means that, with overwhelming probability, for all x, if $e_1(x)$ and $e_2(x)$ have been executed, then $e_3(x)$ has been executed or there exists y such that both $e_4(x, y)$ and $e_5(x, y)$ have been executed.

3.2. Injective Correspondences

Injective correspondences are properties of the form "if some event has been executed n times, then some other events have been executed at least n times". In order to model them in our logical formulae, we extend the grammar of formulae ϕ with injective events $\mathsf{inj\text{-}event}(e(M_1, \ldots, M_m))$. The formula ψ is a conjunction of (injective or non-injective) events. The conditions on the number of executions of events apply only to injective events.

The definition of formula satisfaction is also extended: we indicate at which step each injective event has been executed, by a "pseudo-formula" ϕ^τ obtained from the formula ϕ by replacing terms and non-injective events with $\perp$ and injective events with the step τ at which they have been executed (that is, their index τ in the sequence of events $\mathcal{E}$) or $\perp$ when their execution is not required. For example, if $\phi = \mathsf{inj\text{-}event}(e_1(x)) \wedge (\mathsf{inj\text{-}event}(e_2(x)) \vee \mathsf{inj\text{-}event}(e_3(x)))$, then ϕ^τ is of the form $\tau_1 \wedge (\tau_2 \vee \tau_3)$ where τ_1 is the execution step of $e_1(x)$ and either τ_2 is the execution step of $e_2(x)$ or τ_3 is the execution step of $e_3(x)$. (One of the steps τ_2 and τ_3 may be $\perp$, but not both.) We define formula satisfaction $\rho, \mathcal{E} \vdash^{\phi^\tau} \phi$ as follows:

$\rho, \mathcal{E} \vdash^{\perp} M$ if and only if $\rho, M \Downarrow \mathrm{true}$
$\rho, \mathcal{E} \vdash^{\perp} \mathsf{event}(e(M_1, \ldots, M_m))$ if and only if
for all $j \leq m$, $\rho, M_j \Downarrow a_j$ and $e(a_1, \ldots, a_m) \in \mathcal{E}$
$\rho, \mathcal{E} \vdash^{\tau} \mathsf{inj\text{-}event}(e(M_1, \ldots, M_m))$ if and only if $\tau \neq \perp$,
for all $j \leq m$, $\rho, M_j \Downarrow a_j$, and $e(a_1, \ldots, a_m) = \mathcal{E}(\tau)$
$\rho, \mathcal{E} \vdash^{\phi_1^\tau \wedge \phi_2^\tau} \phi_1 \wedge \phi_2$ if and only if
$\rho, \mathcal{E} \vdash^{\phi_1^\tau} \phi_1$ and $\rho, \mathcal{E} \vdash^{\phi_2^\tau} \phi_2$
$\rho, \mathcal{E} \vdash^{\phi_1^\tau \vee \phi_2^\tau} \phi_1 \vee \phi_2$ if and only if
$\rho, \mathcal{E} \vdash^{\phi_1^\tau} \phi_1$ or $\rho, \mathcal{E} \vdash^{\phi_2^\tau} \phi_2$

This definition differs from the case of non-injective correspondences in that we propagate the pseudo-formula ϕ^τ

and, in the case of injective events, we make sure that the event has been executed at step τ by requiring that $\tau \neq \perp$ and $e(a_1, \dots, a_m) = \mathcal{E}(\tau)$.

Given a function $\mathbb{F}$ that maps ψ^τ to ϕ^τ, the *projection* f of $\mathbb{F}$ to the leaf at occurrence o of ϕ is such that $f(\psi^\tau)$ is the leaf at occurrence o of $\mathbb{F}(\psi^\tau)$. For example, if $\mathbb{F}$ maps ψ^τ to ϕ^τ of the form $\tau_1 \wedge (\tau_2 \vee \tau_3)$, then $\mathbb{F}$ has three projections, which map ψ^τ to τ_1, τ_2, and τ_3 respectively. We say that $\mathbb{F}$ is *component-wise injective* when each projection f of $\mathbb{F}$ is such that $f(\psi_1^\tau) = f(\psi_2^\tau) \neq \perp$ implies $\psi_1^\tau = \psi_2^\tau$. (Ignoring the result $\perp$, f is injective.)

Definition 3 The sequence of events $\mathcal{E}$ *satisfies the correspondence* $\psi \Rightarrow \phi$, written $\mathcal{E} \vdash \psi \Rightarrow \phi$, if and only if there exists a component-wise injective $\mathbb{F}$ such that for all ρ defined on $\mathrm{var}(\psi)$, for all ψ^τ such that $\rho, \mathcal{E} \vdash^{\psi^\tau} \psi$, there exists an extension ρ' of ρ to $\mathrm{var}(\phi)$ such that $\rho', \mathcal{E} \vdash^{\mathbb{F}(\psi^\tau)} \phi$.

Intuitively, a sequence of events $\mathcal{E}$ satisfies $\psi \Rightarrow \phi$ when, if $\mathcal{E}$ satisfies ψ with execution steps defined by ψ^τ, then $\mathcal{E}$ satisfies ϕ with execution steps defined by $\mathbb{F}(\psi^\tau)$. The injectivity is guaranteed because $\mathbb{F}$ is component-wise injective. Definition 2 is unchanged for injective correspondences.

Example 2 Referring to the example G_0 of Section 2, the correspondence

$$\mathsf{inj\text{-}event}(e_B(x,y,z)) \Rightarrow \mathsf{inj\text{-}event}(e_A(x,y,z)) \qquad (2)$$

means that, with overwhelming probability, each execution of $e_B(x,y,z)$ corresponds to a distinct execution of $e_A(x,y,z)$. In this case, ψ^τ is simply the execution step of $e_B(x,y,z)$ and ϕ^τ the execution step of $e_A(x,y,z)$. The function $\mathbb{F}$ is an injective function that maps the execution step of $e_B(x,y,z)$ to the execution step of $e_A(x,y,z)$. (This step is never $\perp$.)

The correspondence

$$\mathsf{event}(e_1(x)) \wedge \mathsf{inj\text{-}event}(e_2(x)) \Rightarrow \mathsf{inj\text{-}event}(e_3(x)) \vee (\mathsf{inj\text{-}event}(e_4(x,y)) \wedge \mathsf{inj\text{-}event}(e_5(x,y)))$$

means that, with overwhelming probability, for all x, if $e_1(x)$ has been executed, then each execution of $e_2(x)$ corresponds to distinct executions of $e_3(x)$ or to distinct executions of $e_4(x,y)$ and $e_5(x,y)$. The function $\mathbb{F}$ maps $\perp \wedge \tau_2$ to $\tau_3 \vee (\tau_4 \wedge \tau_5)$, where τ_2, τ_3, τ_4, τ_5 are the execution steps of $e_2(x)$, $e_3(x)$, $e_4(x,y)$, $e_5(x,y)$ respectively (either τ_3 or τ_4 and τ_5 may be $\perp$). The projections of $\mathbb{F}$ map $\perp \wedge \tau_2$ to τ_3, τ_4, and τ_5 respectively.

When no injective event occurs in $\psi \Rightarrow \phi$, Definition 3 reduces to the definition of non-injective correspondences.

3.3. Property

The next lemma is straightforward. It shows that correspondences are preserved by adding a context.

Lemma 1 *If Q satisfies a correspondence c with public variables V and C is an evaluation context acceptable for (Q, V), then for all $V' \subseteq V \cup (\mathrm{var}(C) \setminus \mathrm{var}(Q))$, $C[Q]$ satisfies c with public variables V'.*

4. Observational Equivalence

The notion of observational equivalence is key to proofs by sequences of games. It can be seen as an adaptation to the computational model of the notion of observational equivalence used in the spi calculus [3] in the Dolev-Yao model. We review the definition observational equivalence and its properties, adapting them to the presence of events.

In the next definition, we use an evaluation context C to represent an algorithm that tries to distinguish Q from Q'.

Definition 4 (Observational equivalence) Let Q and Q' be two processes that satisfy the well-formedness invariants. Let V be a set of variables defined in Q and Q', with the same types.

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, V) and (Q', V), for all channels c and bitstrings a, $|\Pr[C[Q] \rightsquigarrow \overline{c}\langle a \rangle] - \Pr[C[Q'] \rightsquigarrow \overline{c}\langle a \rangle]|$ is negligible and $\sum_{\mathcal{E}} |\Pr[C[Q] \rightsquigarrow \mathcal{E}] - \Pr[C[Q'] \rightsquigarrow \mathcal{E}]|$ is negligible.

This definition formalizes that the probability that an algorithm C distinguishes the games Q and Q' is negligible. The context C is allowed to access directly the variables in V (using find). When V is empty, we write $Q \approx Q'$.

This definition makes events observable, so that observationally equivalent processes execute the same events with overwhelming probability.

The following lemma is straightforward:

Lemma 2 *1. $\approx^V$ is reflexive, symmetric, and transitive.*

2. If $Q \approx^V Q'$ and C is an evaluation context acceptable for (Q, V) and (Q', V), then for all $V' \subseteq V \cup (\mathrm{var}(C) \setminus (\mathrm{var}(Q) \cup \mathrm{var}(Q')))$, $C[Q] \approx^{V'} C[Q']$.

3. If $Q \approx^V Q'$ and Q satisfies a correspondence c with public variables V, then so does Q'.

The transitivity of $\approx^V$ and Property 3 of Lemma 2 are key to performing proofs by sequences of games. Indeed, our prover starts from a game G_0 corresponding to the real protocol, and builds a sequence of observationally equivalent games $G_0 \approx^V G_1 \approx^V \dots \approx^V G_m$. By transitivity, we conclude that $G_0 \approx^V G_m$. By Property 3, if G_m satisfies a certain correspondence with public variables V, then so does G_0. The sequence $G_0 \approx^V G_1 \approx^V \dots \approx^V G_m$ is built by game transformations. Some of these transformations rely on security assumptions of cryptographic primitives; others are syntactic transformations used to simplify games. Since these transformations are the same for correspondences as for secrecy, we do not detail them here, and refer the reader to [20, 21]. (These transformations leave events unchanged.) Next, we illustrate them on an example.

5. A Proof by a Sequence of Games

In this section, we explain the transformations performed on the process G_0 of Section 2. By the unforgeability of signatures, the signature verification with pk_A succeeds only for signatures generated with sk_A. So, when we verify that the signature is correct, we can furthermore check that it has been generated using sk_A. So, after game transformations explained below, we obtain the following final game:

$$
\begin{aligned}
G_1 = {} & c_0(); \mathsf{new}\ rk_A : \mathit{keyseed};\\
& \quad \mathsf{let}\ pk_A = \mathrm{pkgen}'(rk_A)\ \mathsf{in}\ \overline{c_1}\langle pk_A\rangle; (Q_{1A} \mid Q_{1B})\\
Q_{1A} = {} & !^{i_A \leq n} c_2[i_A](x_N : \mathit{nonce}, x_B : \mathit{host});\\
& \quad \mathsf{event}\ e_A(pk_A, x_B, x_N);\\
& \quad \mathsf{let}\ m = \mathrm{concat}(pk_A, x_B, x_N)\ \mathsf{in}\\
& \quad \mathsf{new}\ r : \mathit{seed}; \overline{c_3[i_A]}\langle \mathrm{sign}'(m, \mathrm{skgen}'(rk_A), r)\rangle\\
Q_{1B} = {} & !^{i_B \leq n} c_4[i_B](x_{pk_A} : \mathit{pkey}); \mathsf{new}\ N : \mathit{nonce};\\
& \quad \overline{c_5[i_B]}\langle N, B\rangle; c_6[i_B](s : \mathit{signature});\\
& \quad \mathsf{find}\ u \leq n\ \mathsf{suchthat}\ \mathsf{defined}(m[u], x_B[u], x_N[u])\\
& \quad\ \ \wedge (x_{pk_A} = pk_A) \wedge (B = x_B[u]) \wedge (N = x_N[u])\\
& \quad\ \ \wedge \mathrm{verify}'(\mathrm{concat}(x_{pk_A}, B, N), x_{pk_A}, s)\ \mathsf{then}\\
& \quad \mathsf{event}\ e_B(x_{pk_A}, B, N))
\end{aligned}
$$

The assignment $sk_A = \mathrm{skgen}(rk_A)$ has been removed and $\mathrm{skgen}(rk_A)$ has been substituted for sk_A, in order to make the term $\mathrm{sign}(m, \mathrm{skgen}(rk_A), r)$ appear. This term is needed for the security of the signature scheme to apply.

In Q_{1A}, the signed message is stored in variable m, and this variable is used when computing the signature.

Finally, using the unforgeability of signatures, the signature verification has been replaced with an array lookup: the signature verification can succeed only when $\mathrm{concat}(x_{pk_A}, B, N)$ has been signed with sk_A, so we look for the message $\mathrm{concat}(x_{pk_A}, B, N)$ in the array m and the event e_B is executed only when this message is found. In other words, we look for an index $u \leq n$ such that $m[u]$ is defined and $m[u] = \mathrm{concat}(x_{pk_A}, B, N)$. By definition of m, $m[u] = \mathrm{concat}(pk_A, x_B[u], x_N[u])$, so the equality $m[u] = \mathrm{concat}(x_{pk_A}, B, N)$ can be replaced with $(x_{pk_A} = pk_A) \wedge (B = x_B[u]) \wedge (N = x_N[u])$. (Recall that the result of the concat function contains enough information to recover its arguments.) This transformation replaces the function symbols pkgen, skgen, sign, and verify with primed function symbols pkgen', skgen', sign', and verify' respectively, to avoid repeated applications of the unforgeability of signatures with the same key. (The unforgeability of signatures is applied only to unprimed symbols.)

The soundness of the game transformations shows that $G_0 \approx G_1$. We will prove that G_1 satisfies the correspondences (1) and (2) with any public variables V, in particular with $V = \emptyset$. By Lemma 2, Property 3, G_0 also satisfies these correspondences with public variables $V = \emptyset$. Let us sketch how the proof of correspondence (1) for the game G_1 will proceed. Let Q'_{1A} and Q'_{1B} such that $Q_{1A} = !^{i_A \leq n} Q'_{1A}$ and $Q_{1B} = !^{i_B \leq n} Q'_{1B}$. Assume that event e_B is executed in the copy of Q'_{1B} of index i_B, that is, $e_B(x_{pk_A}[i_B], B, N[i_B])$ is executed. (Recall that the variables x_{pk_A}, N, u, ... are implicitly arrays.) Then the condition of the find above e_B holds, that is, $m[u[i_B]]$, $x_B[u[i_B]]$, and $x_N[u[i_B]]$ are defined, $x_{pk_A}[i_B] = pk_A$, $B = x_B[u[i_B]]$, and $N[i_B] = x_N[u[i_B]]$. Moreover, since $m[u[i_B]]$ is defined, the assignment that defines m has been executed in the copy of Q'_{1A} of index $i_A = u[i_B]$. Then the event $e_A(pk_A, x_B, x_N)$, located above the definition of m, must have been executed in that copy of Q'_{1A}, that is, $e_A(pk_A, x_B[u[i_B]], x_N[u[i_B]])$ has been executed. The equalities in the condition of the find imply that this event is also $e_A(x_{pk_A}[i_B], B, N[i_B])$. To sum up, if $e_B(x_{pk_A}[i_B], B, N[i_B])$ has been executed, then $e_A(x_{pk_A}[i_B], B, N[i_B])$ has been executed, so we have the correspondence (1). This reasoning is typical of the way the prover shows correspondences. In particular, the conditions of array lookups are key in these proofs, because they allow us to relate values in processes that run in parallel (here, the processes that represent A and B), and interesting correspondences relate events that occur in such processes. In the next section, we detail and formalize this reasoning, both for non-injective and injective correspondences.

6. Proving Correspondences

In this section, we explain how our prover shows that a game satisfies a correspondence. We first sketch the technique we use for collecting properties of games, then we handle the simpler case of non-injective correspondences, before generalizing to injective correspondences.

6.1. Reasoning on Games

The proof of correspondences relies on two techniques for reasoning on games. These techniques were already used for simplifying games, so we summarize them briefly and refer the reader to [20] or to Appendix B for details.

First, we collect facts that hold at each program point in the game. We use the following facts: the term M means that M is true, $\mathsf{defined}(M)$ means that M is defined, and $\mathsf{event}(e(M_1, \dots, M_m))$ means that the event $e(M_1, \dots, M_m)$ has been executed. The set of true facts collected at program point P is denoted by $\mathcal{F}_P$. We collect these facts as follows:

- We take into account facts that come from assignments and tests above P. For example, in the process $\mathsf{if}\ M\ \mathsf{then}\ P$, we have $M \in \mathcal{F}_P$, since M is true when P is executed.

  In our running example G_1, at the program point P just after the event e_B, $\mathcal{F}_P$ contains $\mathsf{defined}(m[u[i_B]])$, $\mathsf{defined}(x_B[u[i_B]])$, $\mathsf{defined}(x_N[u[i_B]])$, $x_{pk_A}[i_B] = pk_A$, $B = x_B[u[i_B]]$, and $N[i_B] = x_N[u[i_B]]$, because the condition of find holds when P is executed. ($\mathcal{F}_P$ also contains other facts, which are useless for proving the desired correspondences, so we do not list them.)

- When we already know that $x[\widetilde{M}]$ is defined at P (that is, $\mathsf{defined}(M) \in \mathcal{F}_P$ and $x[\widetilde{M}]$ is a subterm of M), some definition of $x[\widetilde{i}]$ must have been executed, with $\widetilde{i} = \widetilde{M}$, so the facts F that hold at all definitions of x also hold at P, for $\widetilde{i} = \widetilde{M}$: $F\{\widetilde{M}/\widetilde{i}\} \in \mathcal{F}_P$.

  In the example G_1, we have $\mathsf{defined}(m[u[i_B]]) \in \mathcal{F}_P$, and, when $m[i_A]$ is defined, $\mathsf{event}(e_A(pk_A, x_B[i_A], x_N[i_A]))$ holds, so $\mathsf{event}(e_A(pk_A, x_B[i_A], x_N[i_A]))\{u[i_B]/i_A\} \in \mathcal{F}_P$, that is, $\mathsf{event}(e_A(pk_A, x_B[u[i_B]], x_N[u[i_B]])) \in \mathcal{F}_P$. In order words, since m is defined at index $u[i_B]$, event e_A has been executed in the copy of Q'_{1A} of index $u[i_B]$.

Second, we use an equational prover, inspired by the Knuth-Bendix completion algorithm [41]. From a set of facts $\mathcal{F}$, it generates rewrite rules by orienting equalities of $\mathcal{F}$, and uses these rewrite rules to infer new facts from the elements of $\mathcal{F}$. It also takes into account that collisions between uniformly distributed random elements of a large type have negligible probability, so it transforms an equality $x[\widetilde{M}] = x[\widetilde{M'}]$ into $\widetilde{M} = \widetilde{M'}$ when x is defined only by restrictions $\mathsf{new}\ x : T$ and T is a large type. (If the indices were different, the considered cells of x would contain independent random numbers chosen uniformly in the large type T, so the probability of equality would be negligible.)

We say that $\mathcal{F}$ *yields a contradiction* when the equational prover can derive false from $\mathcal{F}$ (for example, when $\mathcal{F}$ contains an inequality $M_1 \neq M_2$, rewritten by the rewrite rules into $M \neq M$, which is then rewritten into false).

6.2. Non-injective Correspondences

Intuitively, in order to prove that a process Q_0 satisfies a non-injective correspondence $\psi \Rightarrow \phi$, we collect all facts that hold at events in ψ and show that these facts imply ϕ using the equational prover.

We collect facts that hold when the event F has been executed, as follows.

Definition 5 (P follows F, $\mathcal{F}_{F,P}$) When $F = \mathsf{event}(e(M_1, \dots, M_m))$ and P is such that $\mathsf{event}\ e(M'_1, \dots, M'_m); P$ occurs in Q_0, we say that P *follows* F, and we define $\mathcal{F}_{F,P} = \theta'\mathcal{F}_P \cup \{\theta' M'_j = M_j \mid j \leq m\}$ where the substitution θ' is a renaming of the replication indices at P to distinct fresh replication indices.

Intuitively, when the event F has been executed, it has been executed by some subprocess of Q_0, so there exists a subprocess $\mathsf{event}\ e(M'_1, \dots, M'_m); P$ in Q_0 such that, for some replication indices defined by θ', the event $e(M'_1, \dots, M'_m)$ has been executed and it is equal to the event F, hence $\theta' M'_j = M_j$ holds for $j \leq m$ and, since the program point P, which follows F, has been reached, $\theta'\mathcal{F}_P$ holds, so $\mathcal{F}_{F,P}$ holds.

Let θ be a substitution equal to the identity on the variables of ψ. This substitution gives values to existentially quantified variables of ϕ. We say that $\mathcal{F} \models\!\!\Rightarrow_\theta \phi$ when we can show that $\mathcal{F}$ implies $\theta\phi$. Formally, we define:

$\mathcal{F} \models\!\!\Rightarrow_\theta M$ if and only if $\mathcal{F} \cup \{\neg\theta M\}$ yields a contradiction

$\mathcal{F} \models\!\!\Rightarrow_\theta \mathsf{event}(e(M_1, \dots, M_m))$ if and only if there exist $M'_1, \dots, M'_m$ such that $\mathsf{event}(e(M'_1, \dots, M'_m)) \in \mathcal{F}$ and $\mathcal{F} \cup \{\bigvee_{j=1}^m \theta M_j \neq M'_j\}$ yields a contradiction

$\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1 \wedge \phi_2$ if and only if $\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1$ and $\mathcal{F} \models\!\!\Rightarrow_\theta \phi_2$

$\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1 \vee \phi_2$ if and only if $\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1$ or $\mathcal{F} \models\!\!\Rightarrow_\theta \phi_2$

Terms θM are proved by contradiction, using the equational prover. Events θF are proved by looking for some event F' in $\mathcal{F}$ and showing by contradiction that $\theta F = F'$, using the equational prover.

Non-injective correspondences are proved as follows.

Proposition 1 *Let $\psi \Rightarrow \phi$ be a non-injective correspondence, with $\psi = F_1 \wedge \ldots \wedge F_m$. If for every P_1 that follows F_1, ..., for every P_m that follows F_m, there exists a substitution θ equal to the identity on the variables of ψ and such that $\mathcal{F}_{F_1,P_1} \cup \ldots \cup \mathcal{F}_{F_m,P_m} \models\!\!\Rightarrow_\theta \phi$, then Q_0 satisfies $\psi \Rightarrow \phi$ with any public variables V.*

Intuitively, when $\psi = F_1 \wedge \ldots \wedge F_m$ holds, $\mathcal{F}_{F_1,P_1} \cup \ldots \cup \mathcal{F}_{F_m,P_m}$ hold. For some θ equal to the identity on ψ, $\mathcal{F}_{F_1,P_1} \cup \ldots \cup \mathcal{F}_{F_m,P_m}$ implies $\theta\phi$, so $\theta\phi$ holds. Hence the correspondence is satisfied. This result is proved in Appendix C.1.

Example 3 Let us prove that the example G_1 satisfies (1). For $\psi = F = \mathsf{event}(e_B(x, y, z))$, the only process P that follows F is the process after event $e_B(x_{pk_A}, B, N)$, so this event has been executed in some copy of Q'_{1B} of index i'_B, with $x_{pk_A}[i'_B] = x, B = y, N[i'_B] = z$. Then, when ψ holds, the facts $\mathcal{F}_{F,P} = \mathcal{F}_P\{i'_B/i_B\} \cup \{x_{pk_A}[i'_B] = x, B = y, N[i'_B] = z\}$ hold for some value of i'_B, where $\mathcal{F}_P$ has been studied in Section 6.1 and $\theta' = \{i'_B/i_B\}$.

Furthermore, the substitution θ is the identity since all variables of ϕ also occur in ψ. Then we just have to show that $\mathcal{F}_{F,P}$ implies $\phi = \mathsf{event}(e_A(x, y, z))$, that is, $\mathcal{F}_{F,P} \models_\theta \mathsf{event}(e_A(x, y, z))$. Since $\mathsf{event}(e_A(pk_A, x_B[u[i_B]], x_N[u[i_B]])) \in \mathcal{F}_P$, we have $\mathsf{event}(e_A(pk_A, x_B[u[i'_B]], x_N[u[i'_B]])) \in \mathcal{F}_{F,P}$, so the equational prover just has to prove by contradiction that $e_A(pk_A, x_B[u[i'_B]], x_N[u[i'_B]]) = e_A(x, y, z)$, that is, $pk_A = x, x_B[u[i'_B]] = y$, and $x_N[u[i'_B]] = z$. The proof succeeds using the following equalities of $\mathcal{F}_{F,P}$: $x_{pk_A}[i'_B] = x$, $B = y$, $N[i'_B] = z$, $x_{pk_A}[i_B] = pk_A$, $B = x_B[u[i'_B]]$, and $N[i'_B] = x_N[u[i'_B]]$.

Hence, G_1 satisfies (1) with any public variables V: if $\psi = \mathsf{event}(e_B(x, y, z))$ has been executed, then $\phi = \mathsf{event}(e_A(x, y, z))$ has been executed.

In the implementation, the substitution θ is initially defined as the identity on $\mathrm{var}(\psi)$. It is defined on other variables when checking $\mathcal{F} \models_\theta M$ by trying to find θ such that $\theta M \in \mathcal{F}$, and when checking $\mathcal{F} \models_\theta \mathsf{event}(e(M_1, \ldots, M_m))$ by trying to find θ such that $\theta\mathsf{event}(e(M_1, \ldots, M_m)) \in \mathcal{F}$. When we do not manage to find the image by θ of all variables of M, resp. $M_1, \ldots, M_m$, the check fails. When there are several suitable facts $\theta M \in \mathcal{F}$ or $\theta\mathsf{event}(e(M_1, \ldots, M_m)) \in \mathcal{F}$, the system tries all possibilities.

6.3. Injective Correspondences

Injective correspondences are more difficult to check than non-injective ones, because they require distinguishing between several executions of the same event. We achieve that as follows.

We require that in the initial game of the sequence, which represents the real protocol, if the event e is used as injective event in a correspondence, then two occurrences of e always occur in different branches of find or if. This property is preserved by the game transformations, so the game Q_0 on which we test the correspondences satisfies this property. This property guarantees that for each value of the replication indices, each injective event is executed at most once.

We add as first argument of every event in Q_0 the tuple $(i_1, \ldots, i_m)$ of replication indices at the program point at which the event is executed. We add as first argument of every event in $\psi \Rightarrow \phi$ a fresh variable. Then the initial process satisfies the initial correspondence if and only if the modified process satisfies the modified correspondence.

The addition of replication indices to events allows us to distinguish executions of the same injective event: these executions always have distinct replication indices by the requirement of the previous paragraph.

We extend Definition 5 to injective events, with exactly the same definition as for non-injective events. We let I_P be the image by θ' of the tuple of replication indices at P, where θ' is the renaming defined in Definition 5.

The proof of injective correspondences extends that for non-injective correspondences: for a correspondence $\psi \Rightarrow \phi$, we additionally prove that distinct executions of the injective events of ψ correspond to distinct executions of each injective event of ϕ, that is, if the injective events of ψ have different replication indices, then each injective event of ϕ has different replication indices. In order to achieve this proof, we collect information on the replication indices of events, for each injective event of ϕ:

- the set of facts $\mathcal{F}$ that are known to hold, which will be used to reason on replication indices of events;
- the replication indices of the considered injective event of ϕ, stored in a tuple M_0; these indices are computed when we prove that this event is executed;
- the replication indices of the injective events of ψ, stored as a mapping $\mathcal{I} = \{j \mapsto I_{P_j} \mid F_j$ is an injective event$\}$, where $\psi = F_1 \wedge \ldots \wedge F_m$ and P_j is the process that executes F_j, for $j \leq m$;
- the set $\mathcal{V}$ containing the replication indices in $\mathcal{F}$ and the variables of ψ; these variables will be renamed to fresh variables in order to avoid conflicts of variable names between different events.

This information is stored in a set $\mathcal{S}$, which contains quadruples $(\mathcal{F}, M_0, \mathcal{I}, \mathcal{V})$. We will show that, if the replication indices of two executions of the injective events of ψ are different, then the replication indices of the corresponding executions of the considered injective event of ϕ are also different. Formally, we consider $(\mathcal{F}, M_0, \mathcal{I}, \mathcal{V})$ and $(\mathcal{F}', M'_0, \mathcal{I}', \mathcal{V}')$ in $\mathcal{S}$. We rename the variables $\mathcal{V}'$ of the second element to fresh variables by a substitution θ'' and show that, if $\mathcal{I} \neq \theta''\mathcal{I}'$, then $M_0 \neq \theta'' M'_0$ (knowing $\mathcal{F}$ and $\theta''\mathcal{F}'$). This property implies injectivity.

Since this reasoning is done for each injective event in ϕ, we collect the associated sets $\mathcal{S}$ in a pseudo-formula $\mathcal{C}$, obtained by replacing each injective event of ϕ with a set $\mathcal{S}$ and all other leaves of ϕ with $\bot$.

We say that $\vdash \mathcal{C}$ when for all non-bottom leaves $\mathcal{S}$ of $\mathcal{C}$, for all $(\mathcal{F}, M_0, \mathcal{I}, \mathcal{V})$, $(\mathcal{F}', M'_0, \mathcal{I}', \mathcal{V}')$ in $\mathcal{S}$, $\mathcal{F} \cup \theta''\mathcal{F}' \cup \{\bigvee_{j \in \mathrm{Dom}(\mathcal{I})} \mathcal{I}(j) \neq \theta''\mathcal{I}'(j), M_0 = \theta'' M'_0\}$ yields a contradiction where the substitution θ'' is a renaming of variables in $\mathcal{V}'$ to distinct fresh variables. As explained above, the condition $\vdash \mathcal{C}$ guarantees injectivity.

We extend the definition of $\mathcal{F} \models_{\theta} \phi$ used for non-injective correspondences to $\mathcal{F} \models_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi$, which means that $\mathcal{F}$ implies $\theta\phi$ and $\mathcal{C}$ correctly collects the tuples $(\mathcal{F}, M_0, \mathcal{I}, \mathcal{V})$ associated to this proof. Formally, we define:

$\mathcal{F} \models_{\theta}^{\mathcal{I},\mathcal{V},\perp} M$ if and only if
$\mathcal{F} \cup \{\neg\theta M\}$ yields a contradiction

$\mathcal{F} \models_{\theta}^{\mathcal{I},\mathcal{V},\perp} \mathsf{event}(e(i, M_1, \dots, M_m))$ if and only if
there exist $M'_0, M'_1, \dots, M'_m$ such that
$\mathsf{event}(e(M'_0, M'_1, \dots, M'_m)) \in \mathcal{F}$ and $\mathcal{F} \cup \{\theta i \neq M'_0 \vee \bigvee_{j=1}^{m} \theta M_j \neq M'_j\}$ yields a contradiction

$\mathcal{F} \models_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{S}} \mathsf{inj\text{-}event}(e(i, M_1, \dots, M_m))$ if and only if
there exist $M'_0, M'_1, \dots, M'_m$ such that
$\mathsf{event}(e(M'_0, M'_1, \dots, M'_m)) \in \mathcal{F}$,
$\mathcal{F} \cup \{\theta i \neq M'_0 \vee \bigvee_{j=1}^{m} \theta M_j \neq M'_j\}$ yields a contradiction, and $(\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}$.

$\mathcal{F} \models_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_1 \wedge \mathcal{C}_2} \phi_1 \wedge \phi_2$ if and only if
$\mathcal{F} \models_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1$ and $\mathcal{F} \models_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_2} \phi_2$

$\mathcal{F} \models_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_1 \vee \mathcal{C}_2} \phi_1 \vee \phi_2$ if and only if
$\mathcal{F} \models_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1$ or $\mathcal{F} \models_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_2} \phi_2$

These formulae differ from the non-injective case in that we propagate $\mathcal{I}$, $\mathcal{V}$, $\mathcal{C}$ and, in the case of injective events, we make sure that quadruples $(\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V})$ are collected correctly by requiring that $(\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}$.

Injective correspondences are proved as follows.

Proposition 2 *Let $\psi \Rightarrow \phi$ be a correspondence, with $\psi = F_1 \wedge \dots \wedge F_m$.*

Assume that, for all events e used as injective events in $\psi \Rightarrow \phi$, two occurrences of the event e always occur in different branches of find *or* if *in Q_0.*

Assume that there exists $\mathcal{C}$ such that $\vdash \mathcal{C}$ and for every P_1 that follows F_1, ..., for every P_m that follows F_m, there exists a substitution θ equal to the identity on the variables of ψ and such that $\mathcal{F}_{F_1,P_1} \cup \dots \cup \mathcal{F}_{F_m,P_m} \models_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi$ where $\mathcal{I} = \{j \mapsto I_{P_j} \mid F_j$ is an injective event$\}$ and $\mathcal{V} = \mathrm{var}(I_{P_1}) \cup \dots \cup \mathrm{var}(I_{P_m}) \cup \mathrm{var}(\psi)$.

Then Q_0 satisfies $\psi \Rightarrow \phi$ with any public variables V.

This result is proved in Appendix C.2. In the implementation, the value of $\mathcal{C}$ is computed by adding $(\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V})$ to $\mathcal{S}$ when handling injective events during the checking of $\mathcal{F}_{F_1,P_1} \cup \dots \cup \mathcal{F}_{F_m,P_m} \models_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi$.

Example 4 Let us prove that the example G_1 satisfies (2). After adding replication indices to events, the process contains events $e_A(i_A, pk_A, x_B, x_N)$ and $e_B(i_B, x_{pk_A}, B, N)$, and we prove the correspondence $\psi \Rightarrow \phi = \mathsf{inj\text{-}event}(e_B(i, x, y, z)) \Rightarrow \mathsf{inj\text{-}event}(e_A(i', x, y, z))$. As in Section 6.1, we compute the set $\mathcal{F}_P$ of facts that hold at the program point P just after event e_B. However, m is defined at index $i_A = u[i_B]$ now implies that $\mathsf{event}(e_A(u[i_B], pk_A, x_B[u[i_B]], x_N[u[i_B]])) \in \mathcal{F}_P$. The process P follows $F = \mathsf{event}(e_B(i, x, y, z))$ and $\mathcal{F} = \mathcal{F}_{F,P} = \mathcal{F}_P\{i'_B/i_B\} \cup \{i'_B = i, x_{pk_A}[i'_B] = x, B = y, N[i'_B] = z\}$.

Similarly to the proof of $\mathcal{F} \models_{\theta} \mathsf{event}(e_A(x, y, z))$ in Example 3, we can show that $\mathcal{F} \models_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}} \mathsf{event}(e_A(i', x, y, z))$ where $\mathcal{I} = \{1 \mapsto i'_B\}$ encodes the replication indices of the events of ψ, $\mathcal{V} = \{i'_B, i, x, y, z\}$ contains the replication indices of $\mathcal{F}$ and the variables of ψ, $\mathcal{C} = \mathcal{S} = \{(\mathcal{F}, u[i'_B], \mathcal{I}, \mathcal{V})\}$. ($\mathcal{C} = \mathcal{S}$ because the formula ψ is reduced to a single event; $M'_0 = u[i_B]$ contains the replication indices of the event e_A contained in $\mathcal{F}$: $\mathsf{event}(e_A(u[i'_B], pk_A, x_B[u[i'_B]], x_N[u[i'_B]])) \in \mathcal{F}$.)

In order to prove injectivity, it remains to show that $\vdash \mathcal{C}$. Let $\theta'' = \{i''_B/i'_B, i''/i, x''/x, y''/y, z''/z\}$. We need to show that $\mathcal{F} \cup \theta''\mathcal{F} \cup \{i'_B \neq i''_B, u[i'_B] = u[i''_B]\}$ yields a contradiction, that is, if the replication indices of the event e_B in ψ are distinct ($i'_B \neq i''_B$), then the replication indices of the event e_A in ϕ are also distinct ($u[i'_B] \neq u[i''_B]$).

$\mathcal{F}$ contains $N[i'_B] = x_N[u[i'_B]]$, so $\theta''\mathcal{F}$ contains $N[i''_B] = x_N[u[i''_B]]$. These two equalities combined with $u[i'_B] = u[i''_B]$ imply that $N[i'_B] = x_N[u[i'_B]] = x_N[u[i''_B]] = N[i''_B]$. Since N is defined by restrictions of the large type *nonce*, $N[i'_B] = N[i''_B]$ implies $i'_B = i''_B$ with overwhelming probability, by eliminating collisions. This equality contradicts $i'_B \neq i''_B$, so we obtain the desired injectivity and G_1 satisfies (2) with any public variables V.

7. Authentication and Key Exchange

In this section, we show how correspondences can be used to prove mutual authentication and authenticated key exchange, as formalized in cryptography following the seminal paper by Bellare and Rogaway [16] and more recent formalizations [7, 27]. Additional discussion and comparisons between these models can be found in Appendix D.

7.1. Mutual Authentication

For simplicity, we consider a protocol that includes two roles, initiator and responder, played by two participants A and B, respectively. Other participants are included in the adversary. The protocol consists of a sequence of messages exchanged alternatively from the initiator to the responder and from the responder to the initiator. Such a configuration can be represented by a process of the form

$$Q_0 = Init; (!^{i_A \leq n} Q_A \mid !^{i_B \leq n} Q_B \mid Q_S)$$

where *Init* is an initialization process (creating keys of A and B for instance), Q_A and Q_B represent respectively the initiator A and the responder B, and Q_S represents a process that allows the adversary to register keys of other (possibly dishonest) participants, so that they can take part in sessions of the protocol with A and B. The processes Q_A and Q_B do not contain replications.

We assume that the protocol contains an odd number of rounds r, so that the first and last messages of the protocol are both from the initiator to the responder. (The other case can be handled similarly.) We assume that the process Q_A stores the messages of the protocol in variables $x_1, \dots, x_r$ and that Q_B stores them in variables $y_1, \dots, y_r$. The initiator process Q_A starts by receiving a message that is not really part of the protocol, and which contains the identity Y of the responder with which A is supposed to run a session. The last (r-th) message sent by process Q_A is assumed to be a pair containing, in addition to the last message of the protocol x_r, either $\mathrm{accept}_A(Y)$, when the protocol ran as expected, or reject, when the protocol failed. The process Q_B is assumed to send a $(r+1)$-th message containing either $\mathrm{accept}_B(X)$ or reject just after B received and checked the last message of the protocol, where X is the identity of its expected partner (inferred by B from the protocol messages). We designate by Q_A^i the copy of Q_A of index $i_A = i$ and by Q_B^i the copy of Q_B of index $i_B = i$. We say that Q_A^i *accepts with* B when it sends $\mathrm{accept}_A(B)$ as second component of its last message; Q_B^i accepts with A when it sends $\mathrm{accept}_B(A)$ as $(r+1)$-th message.

A *session identifier* is a function sid of the protocol messages; $\mathrm{sid}(x_1, \dots, x_r)$ is typically a subsequence of the messages $x_1, \dots, x_r$, often the whole sequence. We also define a partial session identifier $\mathrm{sid}'(x_1, \dots, x_{r-1})$, useful since the r-th message is not available to B when A accepts. We require that $\mathrm{sid}(x_1, \dots, x_r) = \mathrm{sid}(y_1, \dots, y_r)$ implies $\mathrm{sid}'(x_1, \dots, x_{r-1}) = \mathrm{sid}'(y_1, \dots, y_{r-1})$. We say that Q_A^i and $Q_B^{i'}$ are (real) partners when they have the same session identifier: $\mathrm{sid}(x_1[i], \dots, x_r[i]) = \mathrm{sid}(y_1[i'], \dots, y_r[i'])$.

Definition 6 We say that Q_0 is a *secure mutual authentication protocol* with session identifiers sid and sid' if:

1. if the adversary just sends B to Q_A^i as first message and relays messages faithfully between Q_A^i and $Q_B^{i'}$, then Q_A^i accepts with B and $Q_B^{i'}$ accepts with A;
2. with overwhelming probability, there exists an injective function that maps each index i of a process Q_A^i that accepts with B to the index i' of a process $Q_B^{i'}$ with expected partner A such that $\mathrm{sid}'(x_1[i], \dots, x_{r-1}[i]) = \mathrm{sid}'(y_1[i'], \dots, y_{r-1}[i'])$;
3. with overwhelming probability, there exists an injective function that maps each index i' of a process $Q_B^{i'}$ that accepts with A to the index i of a process Q_A^i that accepts with B such that $\mathrm{sid}(x_1[i], \dots, x_r[i]) = \mathrm{sid}(y_1[i'], \dots, y_r[i'])$.

In item 2, $Q_B^{i'}$ has not accepted yet when Q_A^i accepts, so we cannot require that $Q_B^{i'}$ accepts with A; we only require that $Q_B^{i'}$ has expected partner A (so that, if it accepts later, it accepts with A). The first condition is easy to check manually, as already noticed in [16]: it expresses that the protocol works when A and B interact without adversary. The last two conditions mean that each session of A corresponds to a distinct session of B, and conversely, with overwhelming probability. They can be verified using correspondences, as shown by the following proposition.

Proposition 3 *Let Q_0' be obtained from Q_0 by adding*

- $\mathsf{event}\ \ \mathit{part}_A(Y, \mathrm{sid}'(x_1, \dots, x_{r-1}))$; $\mathsf{event}\ \ \mathit{full}_A(Y, \mathrm{sid}(x_1, \dots, x_r))$ *just before A sends $x_r, \mathrm{accept}_A(Y)$;*
- $\mathsf{event}\ \ \mathit{full}_B(X, \mathrm{sid}(y_1, \dots, y_r))$ *just before B sends $\mathrm{accept}_B(X)$;*
- $\mathsf{event}\ \ \mathit{part}_B(X, \mathrm{sid}'(y_1, \dots, y_{r-1}))$ *just before B sends y_{r-1}.*

If Q_0 satisfies the first condition of Definition 6 and Q_0' satisfies the correspondences

$$\mathsf{inj\text{-}event}(\mathit{part}_A(B, x)) \Rightarrow \mathsf{inj\text{-}event}(\mathit{part}_B(A, x)) \quad (3)$$

$$\mathsf{inj\text{-}event}(\mathit{full}_B(A, x)) \Rightarrow \mathsf{inj\text{-}event}(\mathit{full}_A(B, x)) \quad (4)$$

with public variables $V = \emptyset$, then Q_0 is a secure mutual authentication protocol with session identifiers sid and sid'.

The proof of this proposition is straightforward from the definitions. Obviously, many other versions of authentication can be verified using correspondences, for example by requiring non-injective properties instead of injective ones or by requiring authentication in one direction only instead of mutual authentication.

7.2. Authenticated Key Exchange

We adopt the same hypotheses as for mutual authentication. Furthermore, we assume that Q_A sends or receives the j-th message of the protocol on channel $c_{Aj}[i_A]$, and similarly Q_B on channel $c_{Bj}[i_B]$. The channels $c_{Aj}[i_A]$ and $c_{Bj}[i_B]$ are not used for other purposes. We assume that, just before Q_A ends accepting, it stores the established key in variable k_A of type T, and sends $x_r, \mathrm{accept}_A(Y)$ on channel $c_{Ar}[i_A]$. We assume that, just before Q_B ends accepting, it stores the established key in variable k_B of type T and sends $\mathrm{accept}_B(X)$ on channel $c_{Br+1}[i_B]$.

We consider here the Real-Or-Random model [7]: the adversary is allowed to ask several test queries, which either all return the session key (real) or all return a random key (random). Our goal is to show that the adversary has a negligible probability of distinguishing these two situations. As shown in [7], the Real-Or-Random model is stronger than the Find-Then-Guess model of [16]. When the test queries return the real session key, they are defined by the process $Q_T = Q_{TA} \mid Q_{TB}$, where

$$Q_{TA} = !^{i \le n_T} test_A[i](u_A);$$
$$\quad \mathsf{if\ defined}(k_A[u_A])\ \mathsf{then}\ \overline{test_A[i]}\langle k_A[u_A]\rangle$$

and Q_{TB} is defined symmetrically. When the test queries return a random key, they are defined by the process $Q'_T = Q'_{TA} \mid Q'_{TB}$, where

$$\begin{aligned}
&Q'_{TA} = !^{i \le n_T} test_A[i](u_A);\\
&\quad \mathsf{if\ defined}(k_A[u_A], Y[u_A])\ \mathsf{then}\\
&\quad \mathsf{if}\ Y[u_A] \neq B\ \mathsf{then}\ \overline{test_A[i]}\langle k_A[u_A]\rangle\ \mathsf{else}\\
&\quad \mathsf{find}\ u \le n_T\ \mathsf{suchthat\ defined}(u_A[u], r_A[u]) \wedge\\
&\qquad u_A[u] = u_A\ \mathsf{then}\ \overline{test_A[i]}\langle r_A[u]\rangle\ \mathsf{else}\\
&\quad \mathsf{find}\ u \le n_T\ \mathsf{suchthat\ defined}(u_B[u], r_B[u],\\
&\qquad x_1[u_A], \dots, x_r[u_A], y_1[u_B[u]], \dots, y_r[u_B[u]]) \wedge\\
&\qquad \mathrm{sid}(x_1[u_A], \dots, x_r[u_A]) = \mathrm{sid}(y_1[u_B[u]], \dots,\\
&\qquad y_r[u_B[u]])\ \mathsf{then}\ \overline{test_A[i]}\langle r_B[u]\rangle\ \mathsf{else}\\
&\quad \mathsf{new}\ r_A : T; \overline{test_A[i]}\langle r_A\rangle
\end{aligned}$$

and Q'_{TB} is defined symmetrically. When the expected partner of A is not B, the session is executed with a dishonest participant; then, the test query Q'_{TA} returns the real key. When the test query Q'_{TA} has already been asked to the same copy of Q_A (of index $u_A[u] = u_A$), or to a copy of Q_B with the same session identifier (of index $u_B[u]$ such that $\mathrm{sid}(x_1[u_A], \dots, x_r[u_A]) = \mathrm{sid}(y_1[u_B[u]], \dots, y_r[u_B[u]])$), Q'_{TA} returns the same result as in the previous test query. Otherwise, Q'_{TA} returns a fresh random key r_A.

Definition 7 We say that Q_0 is a *secure authenticated key exchange* over T with session identifiers sid and sid' if Q_0 is a secure mutual authentication protocol with session identifiers sid and sid' and the following are true:

1. if the adversary just sends B to Q_A^i as first message and relays messages faithfully between Q_A^i and $Q_B^{i'}$, then Q_A^i accepts with B, $Q_B^{i'}$ accepts with A, $k_A[i] = k_B[i']$, and this random variable is uniformly distributed in T;
2. $Q_0 \mid Q_T \approx Q_0 \mid Q'_T$.

The first point of this definition means that the protocol works correctly when A and B interact without adversary. The second point expresses the indistinguishability when the real key (returned by Q_T) and a random key (returned by Q'_T).

As shown in [20, 21], our prover can prove the secrecy of a variable x, defined as follows:

Definition 8 (Secrecy) Assume x of type T is defined in Q under a single replication $!^{i \le n}$. Let Q' be obtained from Q by removing events. The process Q *preserves the secrecy of* x when $Q' \mid R_x \approx Q' \mid R'_x$, where

$$\begin{aligned}
R_x &= !^{i \le n'} c[i](u : [1, n]); \mathsf{if\ defined}(x[u])\ \mathsf{then}\ \overline{c[i]}\langle x[u]\rangle\\
R'_x &= !^{i \le n'} c[i](u : [1, n]); \mathsf{if\ defined}(x[u])\ \mathsf{then}\\
&\quad \mathsf{find}\ u' \le n'\ \mathsf{suchthat\ defined}(y[u'], u[u']) \wedge u[u'] = u\\
&\quad \mathsf{then}\ \overline{c[i]}\langle y[u']\rangle\ \mathsf{else\ new}\ y : T; \overline{c[i]}\langle y\rangle
\end{aligned}$$

$c \notin \mathrm{fc}(Q')$, and $u, u', y \notin \mathrm{var}(Q')$.

Intuitively, this definition means that the adversary cannot distinguish the array x from an array of uniformly distributed random values by performing several test queries represented by R_x and R'_x, with non-negligible probability.

Proposition 4 *Let Q'_0 be obtained from Q_0 by replacing $\overline{c_{Ar}[i_A]}\langle x_r, \mathrm{accept}_A(Y)\rangle$ with*

$$\begin{aligned}
&\mathsf{event}\ part_A(Y, \mathrm{sid}'(x_1, \dots, x_{r-1}));\\
&\mathsf{event}\ full_A(Y, k_A, \mathrm{sid}(x_1, \dots, x_r));\\
&\mathsf{if}\ Y = B\ \mathsf{then}\\
&\quad \mathsf{let}\ k'_A = k_A\ \mathsf{in}\ \overline{c_{Ar}[i_A]}\langle x_r, \mathrm{accept}_A(Y)\rangle\\
&\mathsf{else}\\
&\quad \overline{c_{Ar}[i_A]}\langle x_r, \mathrm{accept}_A(Y)\rangle; c_{AK}[i_A](); \overline{c_{AK}[i_A]}\langle k_A\rangle
\end{aligned}$$

and $\overline{c_{Br+1}[i_B]}\langle \mathrm{accept}_B(X)\rangle$ with

$$\begin{aligned}
&\mathsf{event}\ full_B(X, k_B, \mathrm{sid}(y_1, \dots, y_r));\\
&\mathsf{if}\ X = A\ \mathsf{then}\\
&\quad \overline{c_{Br+1}[i_B]}\langle \mathrm{accept}_B(X)\rangle\\
&\mathsf{else}\\
&\quad \overline{c_{Br+1}[i_B]}\langle \mathrm{accept}_B(X)\rangle; c_{BK}[i_B](); \overline{c_{BK}[i_B]}\langle k_B\rangle
\end{aligned}$$

and adding event $part_B(X, \mathrm{sid}'(y_1, \dots, y_{r-1}))$ *just before Q_B sends y_{r-1}.*

If Q_0 satisfies the first condition of Definition 7, Q'_0 preserves the secrecy of k'_A, and Q'_0 satisfies the correspondences

$$\mathsf{inj\text{-}event}(part_A(B, x)) \Rightarrow \mathsf{inj\text{-}event}(part_B(A, x)) \tag{5}$$
$$\mathsf{inj\text{-}event}(full_B(A, k, x)) \Rightarrow \mathsf{inj\text{-}event}(full_A(B, k, x)) \tag{6}$$
$$\mathsf{event}(full_B(A, k, x)) \wedge \mathsf{event}(full_A(B, k', x)) \Rightarrow k = k' \tag{7}$$

with public variables $\{k'_A\}$, then Q_0 is a secure authenticated key exchange with session identifiers sid *and* sid'.

This result is proved in Appendix C.3. The process Q'_0 adds events as for mutual authentication, except that the exchanged key is added to the events $full_A$ and $full_B$. Furthermore, when A runs a session with B, it stores the key in the variable k'_A. When A runs a session with $Y \neq B$, it allows the adversary to obtain the exchanged key, by sending a message on c_{AK}, and symmetrically when B runs a session with $X \neq A$. (The test queries also allow the adversary to get the key in this case.) As for Proposition 3, the first condition of Definition 7 is easy to check manually. The first two correspondences imply mutual authentication. The equivalence $Q_0 \mid Q_T \approx Q_0 \mid Q'_T$ is obtained by combining the last two correspondences with the secrecy of k'_A. Intuitively, the correspondences allow us to show that each element of k_B in a session with A is in fact also an element of k'_A (which we can find by looking for the same session identifier), so showing that k'_A cannot be distinguished from an array of independent random numbers is sufficient to show the secrecy of the key. The correspondences must be true with public variables $\{k'_A\}$, so that the context is allowed to access k'_A: in the proof, the process Q'_0 is put in a context that implements the test queries by calling the processes $R_{k'_A}$ or $R'_{k'_A}$ of Definition 8, which directly access k'_A.

8. Experimental Results

We have successfully tested our prover on examples of protocols of the literature: Yahalom [23] with and without key confirmation, Otway-Rees [55], and the original and corrected versions of Woo-Lam shared-key [36] and public-key [62, 64], Needham-Schroeder public-key [48, 53], Denning-Sacco public-key [5, 34], and Needham-Schroeder shared-key [53, 54] with and without key confirmation. For each protocol, we have tried to prove one-way or mutual authentication or authenticated key exchange, depending on the goal of the protocol. Our prover obviously does not prove properties that do not hold. It succeeds in proving properties that hold, in all cases except one: it cannot show (4) for the original version of the Needham-Schroeder shared-key protocol, because it fails to prove that $N_B[i] \neq N_B[i'] - 1$ with overwhelming probability, where N_B is a nonce.

Our prover can make subtle distinctions, which are typically not made by Dolev-Yao provers. For instance, it can model two notions of security for signatures: one in which the adversary is allowed to forge a new signature for an already signed message; the other in which the adversary cannot forge any signature. With the latter definition, for the corrected Woo-Lam public key protocol [64], it can show that the signature is authenticated (both participants have exactly the same signature), while it cannot with the former definition, because the two participants may have different signatures for the same message.

The total runtime for all these tests is 29 s on a Pentium M 1.8 GHz. Appendix E details these results.

9. Conclusion

We have presented the first tool for proving correspondences by sequences of games, in the computational model. This tool works with no or very little help from the user, handles a wide variety of cryptographic primitives, and produces proofs valid for a polynomial number of sessions in the presence of an active adversary.

Although this tool can prove complex correspondences, with conjunctions and disjunctions, our examples use rather simple ones. Complex correspondences proved useful in case studies [1, 2] in the Dolev-Yao model; we plan to use them in similar situations in the computational model. Our tool can also be used to analyze protocols or combinations of primitives that are outside the scope of the Dolev-Yao model. For example, in [22], in collaboration with David Pointcheval, we have used it to prove the Full Domain Hash signature scheme. We plan to consider other such examples in the future.

Acknowledgments We thank David Pointcheval for very helpful discussions on this paper. This work was partly supported by the ANR project ARA SSIA Formacrypt.

References

[1] M. Abadi and B. Blanchet. Computer-assisted verification of a protocol for certified email. *Science of Computer Programming*, 58(1–2):3–27, Oct. 2005. Special issue SAS'03.

[2] M. Abadi, B. Blanchet, and C. Fournet. Just fast keying in the pi calculus. *ACM Transactions on Information and System Security*. To appear. An extended abstract appears in ESOP'04.

[3] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, Jan. 1999. An extended version appeared as Digital Equipment Corporation Systems Research Center report No. 149, January 1998.

[4] M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *TACS'01*, volume 2215 of *LNCS*, pages 82–94. Springer, Oct. 2001.

[5] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, Jan. 1996.

[6] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[7] M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. *IEE Proceedings Information Security*, 153(1):27–39, Mar. 2006.

[8] P. Adão, G. Bana, J. Herzog, and A. Scedrov. Soundness of formal encryption in the presence of key-cycles. In *ESORICS'05*, volume 3679 of *LNCS*, pages 374–396. Springer, Sept. 2005.

[9] M. Backes and P. Laud. Computationally sound secrecy proofs by mechanized flow analysis. In *CCS'06*, pages 370–379. ACM, Nov. 2006.

[10] M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *CSFW'04*, pages 204–218. IEEE, June 2004.

[11] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *CCS'03*, pages 220–230. ACM, Oct. 2003.

[12] M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In *ESORICS'03*, volume 2808 of *LNCS*, pages 271–290. Springer, Oct. 2003.

[13] G. Barthe, J. Cederquist, and S. Tarento. A machine-checked formalization of the generic model and the random oracle model. In *IJCAR'04*, volume 3097 of *LNCS*, pages 385–399. Springer, July 2004.

[14] M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In *ICALP'05*, volume 3580 of *LNCS*, pages 652–663. Springer, July 2005.

[15] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT'00*, volume 1807 of *LNCS*, pages 139–155. Springer, 2000.

[16] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Aug. 1993.

[17] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT'06*, volume 4004 of *LNCS*, pages 409–426. Springer, May 2006. Extended version available at `http://eprint.iacr.org/2004/331`.

[18] B. Blanchet. From secrecy to authenticity in security protocols. In *SAS'02*, volume 2477 of *LNCS*, pages 342–359. Springer, Sept. 2002.

[19] B. Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy*, pages 86–100, May 2004.

[20] B. Blanchet. A computationally sound mechanized prover for security protocols. Cryptology ePrint Archive, Report 2005/401, Nov. 2005. Available at `http://eprint.iacr.org/2005/401`.

[21] B. Blanchet. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy*, pages 140–154, May 2006.

[22] B. Blanchet and D. Pointcheval. Automated security proofs with sequences of games. In *CRYPTO'06*, volume 4117 of *LNCS*, pages 537–554. Springer, Aug. 2006.

[23] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989.

[24] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS'01*, pages 136–145. IEEE, Oct. 2001. An updated version is available at Cryptology ePrint Archive, `http://eprint.iacr.org/2000/067`.

[25] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Linch, O. Pereira, and R. Segala. Time-bounded task-PIOAs: A framework for analyzing security protocols. In *DISC'06*, volume 4167 of *LNCS*, pages 238–253. Springer, Sept. 2006.

[26] R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *TCC'06*, volume 3876 of *LNCS*, pages 380–403. Springer, Mar. 2006. Extended version available at `http://eprint.iacr.org/2004/334`.

[27] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, May 2001.

[28] R. Corin. A probabilistic Hoare-style logic for game-based cryptographic proofs. In *ICALP'06*, volume 4052 of *LNCS*, pages 252–263. Springer, July 2006.

[29] V. Cortier, H. Hördegen, and B. Warinschi. Explicit randomness is not necessary when modeling probabilistic encryption. In *ICS 2006*, Sept. 2006. Proceedings to appear.

[30] V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In *FSTTCS'06*, volume 4246 of *LNCS*, pages 176–187. Springer, Dec. 2006.

[31] V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *ESOP'05*, volume 3444 of *LNCS*, pages 157–171. Springer, Apr. 2005.

[32] A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *ICALP'05*, volume 3580 of *LNCS*, pages 16–29. Springer, July 2005.

[33] A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi. Computationally sound compositional logic for key exchange protocols. In *CSFW'06*, pages 321–334. IEEE Computer Society, July 2006.

[34] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24(8):533–536, Aug. 1981.

[35] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptative chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, Apr. 1988.

[36] A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *CSFW'01*, pages 145–159. IEEE Computer Society, June 2001.

[37] 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`.

[38] J. Herzog. A computational interpretation of Dolev-Yao adversaries. In *WITS'03*, pages 146–155, Apr. 2003.

[39] R. Janvier, Y. Lakhnech, and L. Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *ESOP'05*, volume 3444 of *LNCS*, pages 172–185. Springer, Apr. 2005.

[40] I. R. Jeong, J. Katz, and D. H. Lee. One-round protocols for two-party authenticated key exchange. In *Applied Cryptography and Network Security*, volume 3089 of *LNCS*, pages 220–232. Springer, June 2004.

[41] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[42] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *CRYPTO'05*, volume 3621 of *LNCS*, pages 546–566. Springer, Aug. 2005.

[43] P. Laud. Handling encryption in an analysis for secure information flow. In *ESOP'03*, volume 2618 of *LNCS*, pages 159–173. Springer, Apr. 2003.

[44] P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *IEEE Symposium on Security and Privacy*, pages 71–85, May 2004.

[45] P. Laud. Secrecy types for a simulatable cryptographic library. In *CCS'05*, pages 26–35. ACM, Nov. 2005.

[46] P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *CCS'98*, pages 112–121, Nov. 1998.

[47] P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security protocols. In *FM'99*, volume 1708 of *LNCS*, pages 776–793. Springer, Sept. 1999.

[48] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS'96*, volume 1055 of *LNCS*, pages 147–166. Springer, 1996.

[49] P. Mateus, J. Mitchell, and A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus. In *CONCUR'03*, volume 2761 of *LNCS*, pages 327–349. Springer, Sept. 2003.

[50] D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004.

[51] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *TCC'04*, volume 2951 of *LNCS*, pages 133–151. Springer, Feb. 2004.

[52] J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for the analysis of cryptographic protocols. *Theoretical Computer Science*, 353(1–3):118–164, Mar. 2006.

[53] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, Dec. 1978.

[54] R. M. Needham and M. D. Schroeder. Authentication revisited. *Operating Systems Review*, 21(1):7, 1987.

[55] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.

[56] A. Ramanathan, J. Mitchell, A. Scedrov, and V. Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In *FOSSACS'04*, volume 2987 of *LNCS*, pages 468–483. Springer, Mar. 2004.

[57] V. Shoup. A proposal for an ISO standard for public-key encryption, Dec. 2001. ISO/IEC JTC 1/SC27.

[58] V. Shoup. OAEP reconsidered. *Journal of Cryptology*, 15(4):223–249, Sept. 2002.

[59] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, Nov. 2004. Available at `http://eprint.iacr.org/2004/332`.

[60] C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner. Cryptographically sound theorem proving. In *CSFW'06*, pages 153–166. IEEE, July 2006.

[61] S. Tarento. Machine-checked security proofs of cryptographic signature schemes. In *ESORICS'05*, volume 3679 of *LNCS*, pages 140–158. Springer, Sept. 2005.

[62] T. Y. C. Woo and S. S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, Jan. 1992.

[63] 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, May 1993.

[64] T. Y. C. Woo and S. S. Lam. Authentication for distributed systems. In *Internet Besieged: Countering Cyberspace Scofflaws*, pages 319–355. ACM Press and Addison-Wesley, Oct. 1997.

Appendix

Appendices A, B, and C should be read in this order, because Appendices A and B introduce notations and results used in the proofs in Appendix C.

A. Additional Information on the Calculus

The full syntax of our calculus is given in Figure 1. This calculus distinguishes two categories of processes: input processes wait for a message on a channel; output processes execute some internal computations and output the result on a channel. Most constructs have already been explained in Section 2. We complement these explanations here. The nil process 0 does nothing. The find construct may have several branches: $\mathsf{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ $\mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$ 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. (If there are several successful choices of j, $u_{j1}, \dots, u_{jm_j}$, one of them is chosen randomly with uniform probability.) In case of failure for all branches, it executes P. The conditional $\mathsf{if}\ \mathsf{defined}(M_1, \dots, M_l) \wedge M\ \mathsf{then}\ P\ \mathsf{else}\ P'$ is defined as syntactic sugar for $\mathsf{find}\ \mathsf{suchthat}\ \mathsf{defined}(M_1, \dots, M_l) \wedge M\ \mathsf{then}\ P\ \mathsf{else}\ P'$. The conjunct $\mathsf{defined}(M_1, \dots, M_l)$ can be omitted when $l = 0$ and M can be omitted when it is true. An else branch of find or if may be omitted when it is $\mathsf{else}\ \overline{yield}\langle\rangle; 0$. (Note that "$\mathsf{else}\ 0$" would not be syntactically correct.) A trailing 0 after an output may be omitted.

The semantics of the calculus is formally defined as a probabilistic reduction relation on semantic configurations $\mathbb{C}$. A semantic configuration $\mathbb{C}$ is a quadruple $E, (\sigma, P), \mathcal{Q}, \mathcal{C}$, where

- E is an environment that maps array cells to bitstrings or $\perp$,
- P is the output process currently scheduled and σ is a mapping of the replication indices at P to integers,

$M, N ::=$ 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
- $\mathsf{newChannel}\ c; Q$ channel restriction
- $c[M_1, \ldots, M_l](x_1[\widetilde{i}] : T_1, \ldots, x_k[\widetilde{i}] : T_k); P$ input

$P ::=$ output process
- $\overline{c[M_1, \ldots, M_l]}\langle N_1, \ldots, N_k \rangle; Q$ output
- $\mathsf{new}\ x[\widetilde{i}] : T; P$ random number
- $\mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ P$ assignment
- $\mathsf{if\ defined}(M_1, \ldots, M_l) \wedge M\ \mathsf{then}\ P\ \mathsf{else}\ P'$ conditional
- $\mathsf{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ $\mathsf{suchthat\ defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)$ $\mathsf{else}\ P$ array lookup
- $\mathsf{event}\ e(M_1, \ldots, M_m); P$ event

Figure 1. Syntax of the process calculus

- $\mathcal{Q}$ is a multiset of pairs (σ', Q) where the Q's are input processes currently waiting for messages and σ' is a mapping of the replication indices at Q to integers,
- $\mathcal{C}$ is the set of channels already created.

The semantics is defined by reduction rules of the form $E, (\sigma, P), \mathcal{Q}, \mathcal{C} \xrightarrow{[e]}_{p,t} E', (\sigma', P'), \mathcal{Q}', \mathcal{C}'$ meaning that $E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ reduces to $E', (\sigma', P'), \mathcal{Q}', \mathcal{C}'$ with probability p. The label $[e]$ is empty for all reductions, except events, in which case it records the executed event $e(a_1, \ldots, a_m)$. The tag 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. (Although the semantics depends on the security parameter η, its value is omitted to lighten the notation.)

The semantics uses the relation $E, \sigma, M \Downarrow a$, which means that the term M evaluates to the bitstring a in the environments E (which gives values of arrays) and σ (which gives values of replication indices).

The semantic rule for events is the following:

$$\frac{\forall j \leq m, E, \sigma, M_j \Downarrow a_j}{E, (\sigma, \mathsf{event}\ e(M_1, \ldots, M_m); P), \mathcal{Q}, \mathcal{C} \xrightarrow{e(a_1, \ldots, a_m)}_{1,Ev} E, (\sigma, P), \mathcal{Q}, \mathcal{C}} \quad \text{(Event)}$$

The process evaluates the terms $M_1, \ldots, M_m$ to bitstrings $a_1, \ldots, a_m$, and executes the event $e(a_1, \ldots, a_m)$. This execution is recorded on the label of the transition, and the event instruction disappears from the process. The probability of this transition is 1 and its tag is Ev.

The other semantic rules are the ones of [20], except for minor changes of notations. ([20] used $\xrightarrow{p}_{\eta,t}$ instead of $\xrightarrow{[e]}_{p,t}$ because there was no event. The processes were directly instantiated with the values of the replication indices, so that the semantics of [20] used σP where this paper uses (σ, P).)

The initial configuration for running process Q_0 is $\mathrm{initConfig}(Q_0) = \emptyset, (\sigma_0, \overline{start}\langle\rangle), \{(\sigma_0, Q_0)\}, \mathrm{fc}(Q_0)$ where σ_0 is the function defined nowhere. Hence, the process begins with sending an empty message on channel $start$. The process Q_0 should wait for a message on that channel.

We denote a trace of Q_0 by $\mathrm{initConfig}(Q_0) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ where $p > 0$ is the probability of this trace and $\mathcal{T}$ is a sequence of tags that determine the transitions (one tag per transition).

The following two properties are easy to prove from the definition of the semantics:

Proposition 5 *If* $\mathrm{initConfig}(Q_0) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$, *then P is a subprocess of Q_0 or of* $\overline{start}\langle\rangle.0$.

Proposition 6 *If* $E, (\sigma, P), \mathcal{Q}, \mathcal{C} \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E', (\sigma', P'), \mathcal{Q}', \mathcal{C}'$, *then E' is an extension of E.*

B. Proof Engine

In this section, we define the proof engine that our tool uses for reasoning on games. This engine is used both for simplifying games and for proving correspondences. The version presented here is slightly simplified; the full version can be found in [20]. Our proof engine uses both equations given by the user, that come in particular from algebraic properties of cryptographic primitives, and facts that hold at certain points in the game due to the form of the game. The engine uses this information in order to infer equalities using a Knuth-Bendix-like equational prover.

B.1. User-defined Rewrite Rules

The user can give properties of the form $\forall x_1 : T_1, \ldots, \forall x_m : T_m, M$ which mean that, for all environments ρ that map variables to bitstrings, if for all $j \leq m$, $\rho(x_j) \in I_\eta(T_j)$, then $\rho, M \Downarrow \mathrm{true}$.

These properties are translated into rewrite rules as follows:

- If M is of the form $M_1 = M_2$ and $\mathrm{var}(M_2) \subseteq \mathrm{var}(M_1)$, we 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 $M_1 \neq M_2$, we generate the rewrite rules $\forall x_1 : T_1, \dots, \forall x_m : T_m, (M_1 = M_2) \to \mathrm{false}$, $\forall x_1 : T_1, \dots, \forall x_m : T_m, (M_1 \neq M_2) \to \mathrm{true}$. (Such rules are used for instance to express that different constants are different.)

- Otherwise, we generate the rewrite rule $\forall x_1 : T_1, \dots, \forall x_m : T_m, M \to \mathrm{true}$.

The term M reduces into M' by the rewrite rule $\forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \to M_2$ if and only if $M = C[\theta M_1]$, $M' = C[\theta M_2]$, where C is a term context and θ is a substitution that maps x_j to any term of type T_j for all $j \leq m$.

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

$$\begin{array}{ll}
\neg\mathrm{true} \to \mathrm{false} & \neg\mathrm{false} \to \mathrm{true} \\
\multicolumn{2}{l}{\forall x : \mathit{bool}, \neg(\neg x) \to x} \\
\multicolumn{2}{l}{\forall x : T, \forall y : T, \neg(x = y) \to x \neq y} \\
\multicolumn{2}{l}{\forall x : T, \forall y : T, \neg(x \neq y) \to x = y} \\
\forall x : T, x = x \to \mathrm{true} & \forall x : T, x \neq x \to \mathrm{false} \\
\multicolumn{2}{l}{\forall x : \mathit{bool}, \forall y : \mathit{bool}, \neg(x \wedge y) \to (\neg x) \vee (\neg y)} \\
\multicolumn{2}{l}{\forall x : \mathit{bool}, \forall y : \mathit{bool}, \neg(x \vee y) \to (\neg x) \wedge (\neg y)} \\
\forall x : \mathit{bool}, x \wedge \mathrm{true} \to x & \forall x : \mathit{bool}, x \wedge \mathrm{false} \to \mathrm{false} \\
\forall x : \mathit{bool}, x \vee \mathrm{true} \to \mathrm{true} & \forall x : \mathit{bool}, x \vee \mathrm{false} \to x
\end{array}$$

The prover also has support for commutative function symbols. 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 $\mathrm{true} \wedge M$ into M, $\mathrm{false} \wedge M$ into false, $\mathrm{true} \vee M$ into true, and $\mathrm{false} \vee M$ into M.

B.2. Collecting True Facts from a Game

The function $\mathrm{collectFacts}$ collects facts $\mathsf{defined}(M)$, $\mathsf{event}(e(M_1, \dots, M_m))$, and terms M that hold at each program point of the game. More precisely, for each occurrence P of a subprocess of the game, it computes a set $\mathcal{F}_P$ 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 $\mathrm{collectFacts}$ also computes a set $\mathcal{D}$ containing pairs $(x[\widetilde{i}], P)$ where $x[\widetilde{i}]$ has been defined just above process P. (If there are several definitions of x, there is one such pair for each definition of x.) Finally, for output processes P, $\mathrm{collectFacts}(P)$ returns a set of facts that will hold when the next output is executed, and stores this set in $\mathcal{F}_P^{\mathrm{Fut}}$. (The superscript Fut stands for *future*, since these facts do not hold yet at P, but will hold in the future.)

The function $\mathrm{collectFacts}$ is defined in Figure 2. It is initially called by $\mathrm{collectFacts}(Q_0)$. It takes into account that $x[\widetilde{i}]$ may be defined by an input, a restriction, a let, or a find, and updates $\mathcal{D}$ accordingly. Furthermore, when we execute $\mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ P'$, $x[\widetilde{i}] = M$ holds in P' and $x[\widetilde{i}]$ is defined in P'. When we execute $\mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P'$, 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 $\neg M_j$ holds in P' when $m_j = l_j = 0$. When we execute $\mathsf{event}\ e(M_1, \dots, M_n)$, that execution is recorded by a fact $\mathsf{event}(e(M_1, \dots, M_n))$.

After calling $\mathrm{collectFacts}(Q_0)$, we complete the computed sets $\mathcal{F}_P$ (where P may be an input or output process) by adding facts that come from processes above P:

$$\mathcal{F}_P \leftarrow \mathcal{F}_P \cup \mathcal{F}_{P'} \text{ if } P \text{ is immediately under } P'$$

We also add facts that we can deduce from facts $\mathsf{defined}(M)$. Precisely, if $\mathsf{defined}(M) \in \mathcal{F}_P$, 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 $\mathcal{F}_P$ as follows:

$$\mathcal{F}_P \leftarrow \mathcal{F}_P \cup \left(\bigcap_{(x[i_1,\dots,i_m],P') \in \mathcal{D}} \begin{cases} \sigma(\mathcal{F}_{P'} \cup (\mathcal{F}_{P'}^{\mathrm{Fut}} \cap \mathcal{F}_P)) & \text{if } P \text{ is under } P' \\ \sigma(\mathcal{F}_{P'} \cup \mathcal{F}_{P'}^{\mathrm{Fut}}) & \text{otherwise} \end{cases} \right)$$

where $\sigma = \{M_1/i_1, \dots, M_m/i_m\}$. Indeed, if $\mathsf{defined}(M) \in \mathcal{F}_P$, and $x[M_1, \dots, M_m]$ is a subterm of M, then $x[M_1, \dots, M_m]$ is defined at P, so some definition of $x[M_1, \dots, M_m]$, just above the process P', must have been executed before reaching P, so the facts that hold at P' also hold at P, with a suitable substitution of indices: we have $\sigma\mathcal{F}_{P'}$, that is, $\mathcal{F}_{P'}\{M_1/i_1, \dots, M_m/i_m\}$. Moreover, if the occurrence P is not syntactically under the occurrence P', then the code of P' must have been executed until the next output before executing some other code and reaching P, so in fact $\sigma(\mathcal{F}_{P'} \cup \mathcal{F}_{P'}^{\mathrm{Fut}})$ hold. If P is syntactically under P', it is possible that the code of P' has been executed until reaching P instead of until reaching the next output, so we have only $\sigma(\mathcal{F}_{P'} \cup (\mathcal{F}_{P'}^{\mathrm{Fut}} \cap \mathcal{F}_P))$. If there are several definitions of x, we do not know which one has been executed, so we only add to $\mathcal{F}_P$ the facts that hold in all cases, by taking the intersection on all definitions of x.

This operation may add new defined facts to $\mathcal{F}_P$, 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 $\mathsf{defined}(M)$ in which M contains nested occurrences of the same symbol (such as $x[\dots x[\dots] \dots]$).

$\mathrm{collectFacts}(Q) =$

if $Q = Q_1 \mid Q_2$ then $\mathrm{collectFacts}(Q_1); \mathrm{collectFacts}(Q_2)$

if $Q = !^{i \leq n} Q'$ then $\mathrm{collectFacts}(Q')$

if $Q = \mathsf{newChannel}\ c; Q'$ then $\mathrm{collectFacts}(Q')$

if $Q = c[M_1, \dots, M_l](x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k); P$ then

$\mathcal{F}_P = \{\mathsf{defined}(x_j[\widetilde{i}]) \mid j \leq k\};$

$\mathcal{F}_P^{\mathrm{Fut}} = \mathrm{collectFacts}(P)$

$\mathcal{D} = \mathcal{D} \cup \{(x_j[\widetilde{i}], P) \mid j \leq k\}$

$\mathrm{collectFacts}(P) =$

if $P = \overline{c[M_1, \dots, M_l]}\langle N_1, \dots, N_k \rangle; Q$ then

$\mathrm{collectFacts}(Q); \mathrm{return}\ \emptyset$

if $P = \mathsf{new}\ x[\widetilde{i}] : T; P'$ then

$\mathcal{F}_{P'} = \{\mathsf{defined}(x[\widetilde{i}])\}; \mathcal{F}_{P'}^{\mathrm{Fut}} = \mathrm{collectFacts}(P')$

$\mathcal{D} = \mathcal{D} \cup \{(x[\widetilde{i}], P')\}; \mathrm{return}\ \mathcal{F}_{P'} \cup \mathcal{F}_{P'}^{\mathrm{Fut}}$

if $P = \mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ P'$ then

$\mathcal{F}_{P'} = \{\mathsf{defined}(x[\widetilde{i}]), x[\widetilde{i}] = M\}$

$\mathcal{F}_{P'}^{\mathrm{Fut}} = \mathrm{collectFacts}(P')$

$\mathcal{D} = \mathcal{D} \cup \{(x[\widetilde{i}], P')\}; \mathrm{return}\ \mathcal{F}_{P'} \cup \mathcal{F}_{P'}^{\mathrm{Fut}}$

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

for each $j \leq m$,

$\mathcal{F}_{P_j} = \{\mathsf{defined}(u_{j1}[\widetilde{i'}]), \dots, \mathsf{defined}(u_{jm_j}[\widetilde{i'}]),$
$\mathsf{defined}(M_{j1}), \dots, \mathsf{defined}(M_{jl_j}), M_j\}$

$\mathcal{F}_{P_j}^{\mathrm{Fut}} = \mathrm{collectFacts}(P_j);$

$\mathcal{D} = \mathcal{D} \cup \{(u_{j1}[\widetilde{i'}], P_j), \dots, (u_{jm_j}[\widetilde{i'}], P_j)\}$

$\mathcal{F}_{P'} = \{\neg M_j \mid m_j = l_j = 0\};$

$\mathcal{F}_{P'}^{\mathrm{Fut}} = \mathrm{collectFacts}(P')$

$\mathrm{return}\ (\mathcal{F}_{P'} \cup \mathcal{F}_{P'}^{\mathrm{Fut}}) \cap \bigcap_{j=1}^{m} (\mathcal{F}_{P_j} \cup \mathcal{F}_{P_j}^{\mathrm{Fut}})$

if $P = \mathsf{event}\ e(M_1, \dots, M_n); P'$ then

$\mathcal{F}_{P'} = \{\mathsf{event}(e(M_1, \dots, M_n))\}$

$\mathrm{collectFacts}(P')$

Figure 2. The function $\mathrm{collectFacts}$

We formally define the semantics of facts as follows: $E, \sigma, \mathcal{E} \vdash F$ when the fact F holds in the environments E and σ for the sequence of events $\mathcal{E}$.

$E, \sigma, \mathcal{E} \vdash M$ if and only if $E, \sigma, M \Downarrow \mathrm{true}$
$E, \sigma, \mathcal{E} \vdash \mathsf{defined}(M)$ if and only if
 $E, \sigma, M \Downarrow a$ for some a
$E, \sigma, \mathcal{E} \vdash \mathsf{event}(e(M_1, \dots, M_m))$ if and only if
 there exist $a_1, \dots, a_m$ such that for all $j \leq m$,
 $E, \sigma, M_j \Downarrow a_j$ and $e(a_1, \dots, a_m) \in \mathcal{E}$

We extend this definition to formulae built from facts by conjunctions and disjunctions:

$E, \rho, \mathcal{E} \vdash \phi_1 \wedge \phi_2$ if and only if
 $E, \rho, \mathcal{E} \vdash \phi_1$ and $E, \rho, \mathcal{E} \vdash \phi_2$
$E, \rho, \mathcal{E} \vdash \phi_1 \vee \phi_2$ if and only if
 $E, \rho, \mathcal{E} \vdash \phi_1$ or $E, \rho, \mathcal{E} \vdash \phi_2$

We also extend it naturally to sets of facts and formulae: $E, \sigma, \mathcal{E} \vdash \mathcal{F}$ if and only if for all $F \in \mathcal{F}$, $E, \sigma, \mathcal{E} \vdash F$.

The following proposition expresses the correctness of the collection of true facts. A detailed proof of this result for the full algorithm used in the implementation, but for the version without events, can be found in [20]. The extension to events is straightforward.

Proposition 7 *If* $\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$*, then* $E, \sigma, \mathcal{E} \vdash \mathcal{F}_P$*.*

B.3. Equational Prover

We use an algorithm inspired by the Knuth-Bendix completion algorithm [41], with differences detailed below.

The prover manipulates pairs $\mathcal{F}, \mathcal{R}$ where $\mathcal{F}$ is a set of facts (M, $\mathsf{defined}(M)$, or $\mathsf{event}(e(M_1, \dots, M_n))$) 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 $\mathcal{F}', \mathcal{R}'$):

$$\frac{\mathcal{F} \cup \{F\}, \mathcal{R}}{\mathcal{F} \cup \{F'\}, \mathcal{R}} \quad \text{if } F \text{ reduces into } F' \text{ by a rule of } \mathcal{R} \text{ or a user-defined rewrite rule} \tag{8}$$

$$\frac{\mathcal{F} \cup \{M_1 \wedge M_2\}, \mathcal{R}}{\mathcal{F} \cup \{M_1, M_2\}, \mathcal{R}} \tag{9}$$

$$\frac{\mathcal{F} \cup \{x[M_1, \dots, M_m] = x[M'_1, \dots, M'_m]\}, \mathcal{R}}{\mathcal{F} \cup \{M_1 = M'_1, \dots, M_m = M'_m\}, \mathcal{R}} \tag{10}$$

when x is defined only by restrictions $\mathsf{new}\ x : T$ and T is a large type

$$\frac{\mathcal{F} \cup \{M = M'\}, \mathcal{R}}{\mathcal{F}, \mathcal{R} \cup \{M \to M'\}} \text{ if } M > M' \tag{11}$$

$$\frac{\mathcal{F}, \mathcal{R} \cup \{M_1 \to M_2\}}{\mathcal{F} \cup \{M_1 = M_2'\}, \mathcal{R}} \begin{array}{l}\text{if } M_2 \text{ reduces into } M_2' \text{ by} \\ \text{a rule of } \mathcal{R} \text{ or a} \\ \text{user-defined rewrite rule}\end{array} \tag{12}$$

$$\frac{\mathcal{F}, \mathcal{R} \cup \{M_1 \to M_2\}}{\mathcal{F} \cup \{M_1' = M_2\}, \mathcal{R}} \begin{array}{l}\text{if } M_1 \text{ reduces into } M_1' \text{ by} \\ \text{a rule of } \mathcal{R}\end{array} \tag{13}$$

We also use the symmetric of Rule (11) obtained by swapping the two sides of the equality.

Rule (8) simplifies facts using rewrite rules. Rule (9) decomposes conjunctions of facts. Rule (10) exploits the elimination of collisions between random values. It 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 (11) is applied only when Rules (8) to (10) cannot be applied. Rule (11) 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 only by restrictions, 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 conditions of find or if, and tend to define variables from other variables without creating dependency cycles.)

Rules (12) and (13) are systematically applied to simplify all rewrite rules of $\mathcal{R}$ after a new rewrite rule has been added by Rule (11). Since all terms in rewrite rules of $\mathcal{R}$ are considered as constants, Rule (13) 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 $(\mathcal{F}', \mathcal{R}')$ with $\mathrm{false} \in \mathcal{F}'$.

We write $E, \rho, \mathcal{E} \vdash \mathcal{F}, \mathcal{R}$ when $E, \rho, \mathcal{E} \vdash \mathcal{F}$ and for all $M_1 \to M_2 \in \mathcal{R}$, $E, \rho, \mathcal{E} \vdash M_1 = M_2$. A variant of the following result is proved in [20]. This result shows the soundness of the transformation of $\mathcal{F}, \mathcal{R}$ into $\mathcal{F}', \mathcal{R}'$ for each rule $\frac{\mathcal{F}, \mathcal{R}}{\mathcal{F}', \mathcal{R}'}$ of the equational prover.

Proposition 8 *If* $\frac{\mathcal{F}, \mathcal{R}}{\mathcal{F}', \mathcal{R}'}$, *then* $\Pr[\exists (E, \sigma, P, \mathcal{Q}, \mathcal{C}, \rho, \mathcal{E}), \mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}} E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E, \rho, \mathcal{E} \vdash \mathcal{F}, \mathcal{R} \wedge \neg E, \rho, \mathcal{E} \vdash \mathcal{F}', \mathcal{R}']$ *is negligible.*

We denote by $\Pr[C[Q_0] \rightsquigarrow \mathcal{F}]$ the probability that $C[Q_0]$ reduces into a configuration in which $\mathcal{F}$ holds: $\Pr[C[Q_0] \rightsquigarrow \mathcal{F}] = \Pr[\exists (E, \sigma, P, \mathcal{Q}, \mathcal{C}, \rho, \mathcal{E}), \mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}} E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E, \rho, \mathcal{E} \vdash \mathcal{F}]$.

Proposition 9 *If* $\mathcal{F}$ *yields a contradiction, then* $\Pr[C[Q_0] \rightsquigarrow \mathcal{F}]$ *is negligible.*

Proof This is an easy consequence of Proposition 8. Since $\mathcal{F}$ yields a contradiction, the prover transforms $(\mathcal{F}, \mathcal{R}) = (\mathcal{F}, \emptyset)$ into $(\mathcal{F}', \mathcal{R}')$ that contains false, so $E, \rho, \mathcal{E} \vdash \mathcal{F}$ implies $E, \rho, \mathcal{E} \vdash \mathcal{F}, \mathcal{R}$, and $\neg E, \rho, \mathcal{E} \vdash \mathcal{F}', \mathcal{R}'$. By Proposition 8 applied as many times as there are transformation steps between $(\mathcal{F}, \mathcal{R})$ and $(\mathcal{F}', \mathcal{R}')$, $\Pr[\exists (E, \sigma, P, \mathcal{Q}, \mathcal{C}, \rho, \mathcal{E}), \mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}} E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E, \rho, \mathcal{E} \vdash \mathcal{F}, \mathcal{R} \wedge \neg E, \rho, \mathcal{E} \vdash \mathcal{F}', \mathcal{R}']$ is negligible, which implies that $\Pr[C[Q_0] \rightsquigarrow \mathcal{F}]$ is negligible. $\square$

C. Proofs

C.1. Non-injective Correspondences

The following lemma shows the correctness of $\mathcal{F} \models_\theta \phi$, that is, if $\mathcal{F} \models_\theta \phi$, then $\mathcal{F}$ implies $\theta\phi$ with overwhelming probability.

Lemma 3 *If* $\mathcal{F} \models_\theta \phi$, *then* $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta\phi\}]$ *is negligible.*

Proof The proof proceeds by induction on ϕ.

- Case $\phi = M$. If $\mathcal{F} \cup \{\neg\theta M\}$ yields a contradiction, then, by Proposition 9, $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta\phi\}]$ is negligible.

- Case $\phi = \mathsf{event}(e(M_1, \dots, M_m))$. There are terms $M_1', \dots, M_m'$ such that $\mathsf{event}(e(M_1', \dots, M_m')) \in \mathcal{F}$ and $\mathcal{F} \cup \{\bigvee_{j=1}^m \theta M_j \neq M_j'\}$ yields a contradiction. By Proposition 9, $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\bigvee_{j=1}^m \theta M_j \neq M_j'\}]$ is negligible. Moreover, if $E, \rho, \mathcal{E} \vdash \mathcal{F} \cup \{\neg\theta\phi\}$, then $E, \rho, \mathcal{E} \vdash \mathsf{event}(e(M_1', \dots, M_m'))$ and $\neg E, \rho, \mathcal{E} \vdash \mathsf{event}(e(\theta M_1, \dots, \theta M_m))$, so there exists $j \leq m$ such that $E, \rho, \mathcal{E} \vdash \theta M_j \neq M_j'$, hence $E, \rho, \mathcal{E} \vdash \mathcal{F} \cup \{\bigvee_{j=1}^m \theta M_j \neq M_j'\}$. Therefore, $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta\phi\}] \leq \Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\bigvee_{j=1}^m \theta M_j \neq M_j'\}]$. Hence, $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta\phi\}]$ is negligible.

- Case $\phi = \phi_1 \wedge \phi_2$. We have $\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1$ and $\mathcal{F} \models\!\!\Rightarrow_\theta \phi_2$. By induction hypothesis, $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta\phi_1\}]$ and $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta\phi_2\}]$ are negligible. If $E, \rho, \mathcal{E} \vdash \mathcal{F} \cup \{\neg\theta(\phi_1 \wedge \phi_2)\}$, then either $E, \rho, \mathcal{E} \vdash \mathcal{F} \cup \{\neg\theta\phi_1\}$ or $E, \rho, \mathcal{E} \vdash \mathcal{F} \cup \{\neg\theta\phi_2\}$, so $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta(\phi_1 \wedge \phi_2)\}] \leq \Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta\phi_1\}] + \Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta\phi_2\}]$, so $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta(\phi_1 \wedge \phi_2)\}]$ is negligible.

- Case $\phi = \phi_1 \vee \phi_2$. We have either $\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1$ or $\mathcal{F} \models\!\!\Rightarrow_\theta \phi_2$. In the first case, by induction hypothesis, $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta\phi_1\}]$ is negligible. If $E, \rho, \mathcal{E} \vdash \mathcal{F} \cup \{\neg\theta(\phi_1 \vee \phi_2)\}$, then $E, \rho, \mathcal{E} \vdash \mathcal{F} \cup \{\neg\theta\phi_1\}$, so $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta(\phi_1 \vee \phi_2)\}] \leq \Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta\phi_1\}]$. Therefore, $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\theta(\phi_1 \vee \phi_2)\}]$ is negligible. The second case follows by symmetry. $\square$

Proof of Proposition 1 By hypothesis, if P_1 follows F_1, ..., and P_m follows F_m, then there exists a substitution θ equal to the identity on the variables of ψ and such that $\mathcal{F}_{F_1,P_1} \cup \ldots \cup \mathcal{F}_{F_m,P_m} \models\!\!\Rightarrow_\theta \phi$. We let $\theta(P_1, \ldots P_m)$ be such a substitution and we define $\mathcal{F}(P_1, \ldots, P_m) = \mathcal{F}_{F_1,P_1} \cup \ldots \cup \mathcal{F}_{F_m,P_m} \cup \{\neg\theta\phi\}$ where $\theta = \theta(P_1, \ldots, P_m)$.

Let C be an evaluation context acceptable for (Q_0, V). Below, we show that if $\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ and $\mathcal{E} \not\vdash \psi \Rightarrow \phi$, then there exist P_1 that follows F_1, ..., P_m that follows F_m, and ρ' such that $E, \rho', \mathcal{E} \vdash \mathcal{F}(P_1, \ldots, P_m)$. Therefore,

$$\Pr\left[\begin{array}{l}\exists(E, \sigma, P, \mathcal{Q}, \mathcal{C}, \mathcal{E}),\\ \mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}} E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \mathcal{E} \not\vdash \psi \Rightarrow \phi\end{array}\right]$$
$$\leq \sum_{\substack{P_1,\ldots,P_m \text{ that}\\ \text{follow } F_1,\ldots,F_m \text{ respectively}}} \Pr\left[\begin{array}{l}\exists(E, \sigma, P, \mathcal{Q}, \mathcal{C}, \rho', \mathcal{E}),\\ \mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}\\ \wedge E, \rho', \mathcal{E} \vdash \mathcal{F}(P_1, \ldots, P_m)\end{array}\right]$$
$$\leq \sum_{P_1,\ldots,P_m \text{ that follow } F_1,\ldots,F_m \text{ respectively}} \Pr\left[C[Q_0] \rightsquigarrow \mathcal{F}(P_1, \ldots, P_m)\right]$$

By Lemma 3, since $\mathcal{F}_{F_1,P_1} \cup \ldots \cup \mathcal{F}_{F_m,P_m} \models\!\!\Rightarrow_\theta \phi$ for $\theta = \theta(P_1, \ldots, P_m)$, the probability $\Pr[C[Q_0] \rightsquigarrow \mathcal{F}_{F_1,P_1} \cup \ldots \cup \mathcal{F}_{F_m,P_m} \cup \{\neg\theta\phi\}] = \Pr[C[Q_0] \rightsquigarrow \mathcal{F}(P_1, \ldots, P_m)]$ is negligible, so the sum is negligible since the number of processes $P_1, \ldots, P_m$ is independent of the security parameter. Hence, Q_0 satisfies the correspondence $\psi \Rightarrow \phi$ with public variables V.

Assume that $\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ and for every P_1 that follows F_1, ..., for every P_m that follows F_m, for every ρ', we have $\neg E, \rho', \mathcal{E} \vdash \mathcal{F}(P_1, \ldots, P_m)$. We show that $\mathcal{E} \vdash \psi \Rightarrow \phi$. This result will conclude the proof.

Assume that $\rho, \mathcal{E} \vdash \psi$, where ρ is defined on $\mathrm{var}(\psi)$. For each event $F = \mathsf{event}(e(M_1, \ldots, M_{m'}))$ in ψ, $\rho, \mathcal{E} \vdash \mathsf{event}(e(M_1, \ldots, M_{m'}))$, so for all $j \leq m'$, $\rho, M_j \Downarrow a_j$ and $e(a_1, \ldots, a_{m'}) \in \mathcal{E}$. Since the only transition that produces a label $e(a_1, \ldots, a_{m'})$ is (Event), the trace $\mathrm{initConfig}(Q_0) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ contains a transition of the form $E', (\sigma', \mathsf{event}\ e(M'_1, \ldots, M'_{m'}); P'), \mathcal{Q}', \mathcal{C}' \xrightarrow{e(a_1,\ldots,a_{m'})}_{1,Ev} E', (\sigma', P'), \mathcal{Q}', \mathcal{C}'$ with $E', \sigma', M'_j \Downarrow a_j$ for all $j \leq m'$. By Proposition 5, $\mathsf{event}\ e(M'_1, \ldots, M'_{m'}); P'$ is a subprocess of $C[Q_0]$ or of $\overline{start}\langle\rangle; 0$. Since C does not contain events, $\mathsf{event}\ e(M'_1, \ldots, M'_{m'}); P'$ is a subprocess of Q_0, so P' follows F. By Proposition 7, $E', \sigma', \mathcal{E}' \vdash \mathcal{F}_{P'}$, where $\mathcal{E}'$ is the prefix of $\mathcal{E}$ until and including the considered occurrence of the event $e(a_1, \ldots, a_{m'})$. By Proposition 6, E is an extension of E', so $E, \sigma', \mathcal{E} \vdash \mathcal{F}_{P'}$. Let θ' be the substitution that renames replication indices at P' to fresh replication indices, such that $\mathcal{F}_{F,P'} = \theta'\mathcal{F}_P \cup \{\theta' M'_j = M_j \mid j \leq m'\}$. Let σ'' be such that $\sigma' = \sigma''\theta'$. Then $E, \sigma'', \mathcal{E} \vdash \theta'\mathcal{F}_{P'}$. For all $j \leq m'$, since $E', \sigma', M'_j \Downarrow a_j$, we have $E, \sigma'', \theta' M'_j \Downarrow a_j$. We have $\rho, M_j \Downarrow a_j$. Hence $E, \sigma'' \oplus \rho, \mathcal{E} \vdash \theta' M'_j = M_j$, where $\sigma'' \oplus \rho$ denotes the function that maps x to $\sigma''(x)$ when $x \in \mathrm{Dom}(\sigma'')$ and i to $\rho(i)$ when $i \in \mathrm{Dom}(\rho)$. This function is well defined, since $\mathrm{Dom}(\sigma'')$ and $\mathrm{Dom}(\rho)$ are disjoint. So $E, \sigma'' \oplus \rho, \mathcal{E} \vdash \mathcal{F}_{F,P'}$.

Therefore, for each F_j in ψ, there exist σ''_j, ρ, and a process P_j that follows F_j such that $E, \sigma''_j \oplus \rho, \mathcal{E} \vdash \mathcal{F}_{F_j,P_j}$. Since the environments σ''_j and ρ have disjoint domains, we can define an environment $\rho' = \sigma''_1 \oplus \ldots \oplus \sigma''_m \oplus \rho$. Then $E, \rho', \mathcal{E} \vdash \mathcal{F}_{F_1,P_1} \cup \ldots \cup \mathcal{F}_{F_m,P_m}$.

Let $\theta = \theta(P_1, \ldots, P_m)$. Since $E, \rho', \mathcal{E} \vdash \mathcal{F}_{F_1,P_1} \cup \ldots \cup \mathcal{F}_{F_m,P_m}$ and $\neg E, \rho', \mathcal{E} \vdash \mathcal{F}(P_1, \ldots, P_m)$, we have $E, \rho', \mathcal{E} \vdash \theta\phi$. We extend ρ to all $x \in \mathrm{var}(\phi) \setminus \mathrm{var}(\psi)$, in such a way that $E, \rho', \theta(x) \Downarrow \rho(x)$. Moreover, if $x \in \mathrm{var}(\psi)$, then $\rho(x) = \rho(\theta(x)) = \rho'(\theta(x))$ since $\theta x = x$, so $E, \rho', \theta(x) \Downarrow \rho(x)$. So, for all $x \in \mathrm{var}(\phi)$, $E, \rho', \theta(x) \Downarrow \rho(x)$. Since $E, \rho', \mathcal{E} \vdash \theta\phi$, we have $\rho, \mathcal{E} \vdash \phi$, so $\mathcal{E}$ satisfies the correspondence $\psi \Rightarrow \phi$. $\square$

C.2. Injective Correspondences

We define $\mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)$ as follows:

$$\mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\perp} M) = \theta M$$
$$\mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\perp} \mathsf{event}(e(M_0, \ldots, M_m))) = \theta\mathsf{event}(e(M_0, \ldots, M_m))$$
$$\mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{S}} \mathsf{inj\text{-}event}(e(M_0, \ldots, M_m))) = \bigvee_{\mathsf{event}(e(M'_0,\ldots,M'_m)) \in \mathcal{F} \wedge (\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}} \left(\bigwedge_{j=0}^m \theta M_j = M'_j\right)$$
$$\mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1 \wedge \mathcal{C}_2} \phi_1 \wedge \phi_2) = \mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1) \wedge \mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_2} \phi_2)$$

$$\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1 \vee \mathcal{C}_2} \phi_1 \vee \phi_2) = \mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1) \vee \mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_2} \phi_2)$$

where M_0 is a fresh variable added as first argument of events. The formula $\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)$ generalizes $\theta\phi$ to the case of injective events.

The next lemma shows that, if $\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi$, then $\mathcal{F}$ implies $\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)$ with overwhelming probability.

Lemma 4 *If $\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi$, then $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)\}]$ is negligible.*

Proof The proof is similar to that of Lemma 3, and proceeds by induction on ϕ. The only new case is the one of injective events.

- Case $\phi = \mathsf{inj\text{-}event}(e(M_0, \dots, M_m))$, $\mathcal{C} = \mathcal{S}$. There are terms $M'_0, \dots, M'_m$ such that $\mathsf{event}(e(M'_0, \dots, M'_m)) \in \mathcal{F}$, $\mathcal{F} \cup \{\bigvee_{j=0}^m \theta M_j \neq M'_j\}$ yields a contradiction, and $(\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}$. By Proposition 9, $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\theta \bigvee_{j=0}^m \theta M_j \neq M'_j\}]$ is negligible. Moreover, if $E, \rho, \mathcal{E} \vdash \mathcal{F} \cup \{\neg\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{S}} \phi)\}$, then $E, \rho, \mathcal{E} \vdash \mathcal{F}$ and for all $M'_0, \dots, M'_m$ such that $\mathsf{event}(e(M'_0, \dots, M'_m)) \in \mathcal{F}$ and $(\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}$, in particular for the terms $M'_0, \dots, M'_m$ above, we have $E, \rho, \mathcal{E} \vdash \bigvee_{j=0}^m \theta M_j \neq M'_j$, so $E, \rho, \mathcal{E} \vdash \mathcal{F} \cup \{\bigvee_{j=0}^m \theta M_j \neq M'_j\}$. Therefore, $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{S}} \phi)\}] \leq \Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\bigvee_{j=0}^m \theta M_j \neq M'_j\}]$. Hence, $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{S}} \phi)\}]$ is negligible. $\square$

The next lemma details the meaning of $\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)$. Essentially, this formula implies $\theta\phi$, so, if we store in $\rho(x)$ the value of $\theta(x)$ by $E, \rho', \theta(x) \Downarrow \rho(x)$, we have $\rho, \mathcal{E} \vdash^{\phi^\tau} \phi$. Furthermore, for injective events, $\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)$ guarantees that the quadruples $(\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V})$ are correctly collected in $\mathcal{C}$.

Lemma 5 *If $E, \rho', \mathcal{E} \vdash \mathcal{F}$, for all $x \in \mathrm{var}(\phi)$, $E, \rho', \theta(x) \Downarrow \rho(x)$, and $E, \rho', \mathcal{E} \vdash \mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)$, then there exists ϕ^τ such that $\rho, \mathcal{E} \vdash^{\phi^\tau} \phi$ and, if τ is a non-bottom leaf of ϕ^τ and $\mathcal{S}$ the corresponding leaf of $\mathcal{C}$, then $E, \rho', \mathcal{E} \vdash \mathcal{E}(\tau) = \mathsf{event}(e(M'_0, \dots, M'_m))$ for some $\mathsf{event}(e(M'_0, \dots, M'_m)) \in \mathcal{F}$ and $(\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}$.*

Proof The proof proceeds by induction on ϕ.

- Case $\phi = M$. We have $\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi) = \theta M$, so $E, \rho', \mathcal{E} \vdash \theta M$, so $\rho, M \Downarrow \mathrm{true}$, so $\rho, \mathcal{E} \vdash^\perp M$. The result holds with $\phi^\tau = \perp$.
- Case $\phi = \mathsf{event}(e(M_0, \dots, M_m))$. We have $\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi) = \theta\mathsf{event}(e(M_0, \dots, M_m))$, so $E, \rho', \mathcal{E} \vdash \theta\mathsf{event}(e(M_0, \dots, M_m))$, so $\rho, \mathcal{E} \vdash^\perp \mathsf{event}(e(M_0, \dots, M_m))$. The result holds with $\phi^\tau = \perp$.
- Case $\phi = \mathsf{inj\text{-}event}(e(M_0, \dots, M_m))$. We have $\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{S}} \phi) = \bigvee_{\mathsf{event}(e(M'_0, \dots, M'_m)) \in \mathcal{F} \wedge (\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}} \left(\bigwedge_{j=0}^m \theta M_j = M'_j\right)$. So there exist $M'_0, \dots, M'_m$ such that $\mathsf{event}(e(M'_0, \dots, M'_m)) \in \mathcal{F}$, $(\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}$, and $E, \rho', \mathcal{E} \vdash \bigwedge_{j=0}^m \theta M_j = M'_j$. We have $E, \rho', \mathcal{E} \vdash \mathcal{F}$, so $E, \rho', \mathcal{E} \vdash \mathsf{event}(e(M'_0, \dots, M'_m))$, so $E, \rho', \mathcal{E} \vdash \theta\mathsf{event}(e(M_0, \dots, M_m))$, so there exists τ such that $\mathcal{E}(\tau) = \mathsf{event}(e(a_0, \dots, a_m))$ with for all $j \leq m$, $E, \rho', \theta M_j \Downarrow a_j$, so for all $j \leq m$, $E, \rho, M_j \Downarrow a_j$, so $\rho, \mathcal{E} \vdash^\tau \mathsf{event}(M_0, \dots, M_m))$. Moreover, $E, \rho', \mathcal{E} \vdash \mathcal{E}(\tau) = \mathsf{event}(e(a_0, \dots, a_m)) = \theta\mathsf{event}(e(M_0, \dots, M_m)) = \mathsf{event}(e(M'_0, \dots, M'_m))$. As already noticed, we have $\mathsf{event}(e(M'_0, \dots, M'_m)) \in \mathcal{F}$ and $(\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}$, so the result holds with $\phi^\tau = \tau$.
- Case $\phi = \phi_1 \wedge \phi_2$. We have $E, \rho', \mathcal{E} \vdash \mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1 \wedge \mathcal{C}_2} \phi_1 \wedge \phi_2)$, so $E, \rho', \mathcal{E} \vdash \mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1)$ and $E, \rho', \mathcal{E} \vdash \mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_2} \phi_2)$. The induction hypothesis yields ϕ_1^τ and ϕ_2^τ, and the result holds with $\phi^\tau = \phi_1^\tau \wedge \phi_2^\tau$.
- Case $\phi = \phi_1 \vee \phi_2$. We have $E, \rho', \mathcal{E} \vdash \mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1 \vee \mathcal{C}_2} \phi_1 \vee \phi_2)$, so $E, \rho', \mathcal{E} \vdash \mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1)$ or $E, \rho', \mathcal{E} \vdash \mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_2} \phi_2)$. In the first case, the induction hypothesis yields ϕ_1^τ, and the result holds with $\phi^\tau = \phi_1^\tau \vee \phi_2^\perp$, where $\phi_2^\perp$ is the formula ϕ_2 in which all terms and events have been replaced with $\perp$. The second case follows by symmetry. $\square$

The next lemma shows that, for events e used as injective events, two distinct executions of event e have distinct replication indices. This is a consequence of the requirement that two occurrences of the same event e be in different branches of find or if in Q_0.

When the term M contains no array accesses, we define $\sigma(M)$ by $E, \sigma, M \Downarrow \sigma(M)$ for any environment E, since the evaluation of M does not depend on E.

Lemma 6 *Assume that the event e is used as injective event in the correspondence $\psi \Rightarrow \phi$. If the trace $\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} \mathbb{C}$ contains two distinct reduc-*

tions

$$E, (\sigma, \mathsf{event}\ e(M_0, \dots, M_m); P), \mathcal{Q}, \mathcal{C} \xrightarrow{e(a_0,\dots,a_m)}_{1,Ev} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$$

and $E', (\sigma', \mathsf{event}\ e(M'_0, \dots, M'_m); P'), \mathcal{Q}', \mathcal{C}' \xrightarrow{e(a'_0,\dots,a'_m)}_{1,Ev} E', (\sigma', P'), \mathcal{Q}', \mathcal{C}'$

then $a_0 \neq a'_0$.

Proof Let us fix the event symbol e. We define $\mathit{Events}(\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} \mathbb{C})$ as the multiset that contains a_0 for each reduction $E, (\sigma, \mathsf{event}\ e(M_0, \dots, M_m); P), \mathcal{Q}, \mathcal{C} \xrightarrow{e(a_0,\dots,a_m)}_{1,Ev} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ in the trace $\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} \mathbb{C}$. Multisets S are represented by functions that map each element x of S to the number of occurrences of x in S. When S_1 and S_2 are multisets, the multiset $\max(S_1, S_2)$ is defined by $\max(S_1, S_2)(x) = \max(S_1(x), S_2(x))$. We define the multisets $\mathit{Events}(\sigma, P)$ and $\mathit{Events}(\sigma, Q)$ by

$$\mathit{Events}(\sigma, 0) = \emptyset$$
$$\mathit{Events}(\sigma, Q_1 \mid Q_2) = \mathit{Events}(\sigma, Q_1) \uplus \mathit{Events}(\sigma, Q_2)$$
$$\mathit{Events}(\sigma, !^{i \leq n} Q) = \biguplus_{a \in [1, I_\eta(n)]} \mathit{Events}(\sigma[i \mapsto a], Q)$$
$$\mathit{Events}(\sigma, \mathsf{newChannel}\ c; Q) = \mathit{Events}(\sigma, Q)$$
$$\mathit{Events}(\sigma, c[M_1, \dots, M_l](x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k); P) = \mathit{Events}(\sigma, P)$$
$$\mathit{Events}(\sigma, \overline{c[M_1, \dots, M_l]}\langle N_1, \dots, N_k\rangle; Q) = \mathit{Events}(\sigma, Q)$$
$$\mathit{Events}(\sigma, \mathsf{new}\ x[\widetilde{i}] : T; P) = \mathit{Events}(\sigma, P)$$
$$\mathit{Events}(\sigma, \mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ P) = \mathit{Events}(\sigma, P)$$
$$\mathit{Events}(\sigma, \mathsf{event}\ e'(M_0, \dots, M_m); P) = \mathit{Events}(\sigma, P) \text{ if } e' \neq e$$
$$\mathit{Events}(\sigma, \mathsf{event}\ e(M_0, \dots, M_m); P) = \{\sigma(M_0)\} \uplus \mathit{Events}(\sigma, P)$$
$$\mathit{Events}(\sigma, \mathsf{find}\ (\textstyle\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P) = \max(\max_{j=1}^{m} \mathit{Events}(\sigma, P_j), \mathit{Events}(\sigma, P))$$

We define the multiset $\mathit{Events}(E, (\sigma, P), \mathcal{Q}, \mathcal{C}) = \mathit{Events}(\sigma, P) \uplus \biguplus_{(\sigma', Q') \in \mathcal{Q}} \mathit{Events}(\sigma', Q')$. This multiset contains all bitstrings a_0 for events $e(\dots)$ that may be executed in a trace that begins with $E, (\sigma, P), \mathcal{Q}, \mathcal{C}$.

The multiset $\mathit{Events}(\mathrm{initConfig}(C[Q_0]))$ contains no duplicates, since two occurrences of the same event e must be in different branches of find or if in Q_0 and C does not contain events. Moreover, for the empty trace ϵ, $\mathit{Events}(\epsilon) = \emptyset$, so $\mathit{Events}(\epsilon) \uplus \mathit{Events}(\mathrm{initConfig}(C[Q_0]))$ contains no duplicates.

We show that, if $\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} \mathbb{C} \xrightarrow{[e]}_{p',t'} \mathbb{C}'$, then $\mathit{Events}(\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} \xrightarrow{[e]}_{p',t'} \mathbb{C}') \uplus \mathit{Events}(\mathbb{C}') \subseteq \mathit{Events}(\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} \mathbb{C}) \uplus \mathit{Events}(\mathbb{C})$.

Thus, if $\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} \mathbb{C}$, then

$$\begin{aligned}&\mathit{Events}(\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} \mathbb{C}) \uplus \mathit{Events}(\mathbb{C})\\ &\quad\subseteq \mathit{Events}(\epsilon) \uplus \mathit{Events}(\mathrm{initConfig}(C[Q_0]))\end{aligned}$$

These multisets contain no duplicates, so in particular,

$$\mathit{Events}(\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} \mathbb{C})$$

contains no duplicates. This property implies the desired result. $\square$

Proof of Proposition 2 By hypothesis, if P_1 follows F_1, ..., and P_m follows F_m, then there exists a substitution θ equal to the identity on the variables of ψ and such that $\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi$ where $\mathcal{F} = \mathcal{F}_{F_1,P_1} \cup \ldots \cup \mathcal{F}_{F_m,P_m}$, $\mathcal{I} = \{j \mapsto I_{P_j} \mid F_j \text{ is an injective event}\}$, and $\mathcal{V} = \mathrm{var}(I_{P_1}) \cup \ldots \cup \mathrm{var}(I_{P_m}) \cup \mathrm{var}(\psi)$. We let $\theta(P_1, \ldots P_m)$ be such a substitution and we define $\mathcal{F}(P_1, \ldots, P_m) = \mathcal{F} \cup \{\neg\mathrm{formula}(\mathcal{F} \models_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)\}$ where $\theta = \theta(P_1, \ldots, P_m)$.

Let C be an evaluation context acceptable for (Q_0, V). Next, we show that if $\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ and $\mathcal{E} \not\vdash \psi \Rightarrow \phi$, then

- there exist P_1 that follows $F_1, \ldots, P_m$ that follows F_m, and ρ' such that $E, \rho', \mathcal{E} \vdash \mathcal{F}(P_1, \ldots, P_m)$,
- or there exist a non-bottom leaf $\mathcal{S}$ of $\mathcal{C}$, $(\mathcal{F}, M_0, \mathcal{I}, \mathcal{V})$ and $(\mathcal{F}', M'_0, \mathcal{I}', \mathcal{V}')$ in $\mathcal{S}$, and ρ' such that $E, \rho', \mathcal{E} \vdash \mathcal{F} \cup \mathcal{F}' \cup \{\bigvee_{j \in \mathrm{Dom}(\mathcal{I})} \mathcal{I}(j) \neq \theta''\mathcal{I}'(j)\} \cup \{\theta'' M'_0 = M_0\}$, where the substitution θ'' is a renaming of the variables in $\mathcal{V}'$ to distinct fresh variables.

Therefore,

$$\begin{aligned}
&\Pr\left[\begin{array}{l}\exists(E, \sigma, P, \mathcal{Q}, \mathcal{C}, \mathcal{E}),\\ \mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}} E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \mathcal{E} \not\vdash \psi \Rightarrow \phi\end{array}\right]\\
&\leq \sum_{\substack{P_1,\ldots,P_m \text{ that}\\ \text{follow } F_1,\ldots,F_m \text{ respectively}}} \Pr\left[\begin{array}{l}\exists(E, \sigma, P, \mathcal{Q}, \mathcal{C}, \rho', \mathcal{E}),\\ \mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}\\ \wedge E, \rho', \mathcal{E} \vdash \mathcal{F}(P_1, \ldots, P_m)\end{array}\right]\\
&+ \sum_{\substack{\mathcal{S} \text{ leaf of } \mathcal{C}, \mathcal{S} \neq \bot,\\ (\mathcal{F}, M_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S},\\ (\mathcal{F}', M'_0, \mathcal{I}', \mathcal{V}') \in \mathcal{S}}} \Pr\left[\begin{array}{l}\exists(E, \sigma, P, \mathcal{Q}, \mathcal{C}, \rho', \mathcal{E}),\\ \mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}\\ \wedge E, \rho', \mathcal{E} \vdash \mathcal{F} \cup \mathcal{F}' \cup \{M_0 = \theta'' M'_0\}\\ \quad \cup \{\bigvee_{j \in \mathrm{Dom}(\mathcal{I})} \mathcal{I}(j) \neq \theta''\mathcal{I}'(j)\}\end{array}\right]
\end{aligned}$$

$$\leq \sum_{P_1,\dots,P_m \text{ that follow } F_1,\dots,F_m \text{ respectively}} \Pr\left[C[Q_0] \rightsquigarrow \mathcal{F}(P_1,\dots,P_m)\right]$$

$$+ \sum_{\substack{\mathcal{S} \text{ leaf of } \mathcal{C}, \mathcal{S} \neq \bot,\\ (\mathcal{F}, M_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}, (\mathcal{F}', M_0', \mathcal{I}', \mathcal{V}') \in \mathcal{S}}} \Pr\begin{bmatrix} C[Q_0] \rightsquigarrow \mathcal{F} \cup \mathcal{F}' \cup \{M_0 = \theta'' M_0'\} \\ \cup \{\bigvee_{j \in \mathrm{Dom}(\mathcal{I})} \mathcal{I}(j) \neq \theta'' \mathcal{I}'(j)\} \end{bmatrix}$$

By Lemma 4, since $\mathcal{F} \models\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi$, the probability $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \{\neg\mathrm{formula}(\mathcal{F} \models\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)\}]$ is negligible, that is, $\Pr\left[C[Q_0] \rightsquigarrow \mathcal{F}(P_1,\dots,P_m)\right]$ is negligible. Since $\vdash \mathcal{C}$, for all non-bottom leaves $\mathcal{S}$ of $\mathcal{C}$, for all $(\mathcal{F}, M_0, \mathcal{I}, \mathcal{V})$, $(\mathcal{F}', M_0', \mathcal{I}', \mathcal{V}')$ in $\mathcal{S}$, $\mathcal{F} \cup \theta''\mathcal{F}' \cup \{\bigvee_{j \in \mathrm{Dom}(\mathcal{I})} \mathcal{I}(j) \neq \theta''\mathcal{I}'(j), M_0 = \theta'' M_0'\}$ yields a contradiction. By Proposition 9, $\Pr[C[Q_0] \rightsquigarrow \mathcal{F} \cup \mathcal{F}' \cup \{M_0 = \theta'' M_0'\} \cup \{\bigvee_{j \in \mathrm{Dom}(\mathcal{I})} \mathcal{I}(j) \neq \theta''\mathcal{I}'(j)\}]$ is negligible. Hence the sum is negligible, so Q_0 satisfies the correspondence $\psi \Rightarrow \phi$ with public variables V.

Assume that

- $\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$,
- for every P_1 that follows F_1, …, for every P_m that follows F_m, for every ρ', we have $\neg E, \rho', \mathcal{E} \vdash \mathcal{F}(P_1,\dots,P_m)$,
- and for every non-bottom leaf $\mathcal{S}$ of $\mathcal{C}$, for every $(\mathcal{F}, M_0, \mathcal{I}, \mathcal{V})$ and $(\mathcal{F}', M_0', \mathcal{I}', \mathcal{V}')$ in $\mathcal{S}$, for every ρ', we have $\neg E, \rho', \mathcal{E} \vdash \mathcal{F} \cup \mathcal{F}' \cup \{\bigvee_{j \in \mathrm{Dom}(\mathcal{I})} \mathcal{I}(j) \neq \theta''\mathcal{I}'(j)\} \cup \{M_0 = \theta'' M_0'\}$, where the substitution θ'' is a renaming of the variables in $\mathcal{V}'$ to distinct fresh variables.

We show that $\mathcal{E} \vdash \psi \Rightarrow \phi$.

Assume that $\rho, \mathcal{E} \vdash^{\psi^\tau} \psi$, where ρ is defined on $\mathrm{var}(\psi)$, $\psi = F_1 \wedge \dots \wedge F_m$, $\psi^\tau = \tau_1 \wedge \dots \wedge \tau_m$, and for all $j \leq m$, τ_j is either a step or $\bot$. For each event $F_j = \mathsf{event}(e_j(M_{j0},\dots,M_{jm_j}))$ or $F_j = \mathsf{inj\text{-}event}(e_j(M_{j0},\dots,M_{jm_j}))$ in ψ, we have $\rho, \mathcal{E} \vdash^{\tau_j} \mathsf{event}(e_j(M_{j0},\dots,M_{jm_j}))$, so $\rho, M_{jk} \Downarrow a_{jk}$ for all $k \leq m_j$ and $e_j(a_{j0},\dots,a_{jm_j}) \in \mathcal{E}$. Moreover, if $F_j = \mathsf{inj\text{-}event}(e_j(M_{j0},\dots,M_{jm_j}))$, then $e_j(a_{j0},\dots,a_{jm_j}) = \mathcal{E}(\tau_j)$. Since the only transition that produces a label $e_j(a_{j0},\dots,a_{jm_j})$ is (Event), the trace $\mathrm{initConfig}(Q_0) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ contains a transition of the form $E_j, (\sigma_j, \mathsf{event}\ e_j(M_{j0}',\dots,M_{jm_j}'); P_j), \mathcal{Q}_j, \mathcal{C}_j \xrightarrow{e_j(a_{j0},\dots,a_{jm_j})}_{1,Ev} E_j, (\sigma_j, P_j), \mathcal{Q}_j, \mathcal{C}_j$ with $E_j, \sigma_j, M_{jk}' \Downarrow a_{jk}$ for all $k \leq m_j$. By Proposition 5, $\mathsf{event}\ e_j(M_{j0}',\dots,M_{jm_j}'); P_j$ is a subprocess of $C[Q_0]$ or of $\overline{start}\langle\rangle; 0$. Since C does not contain events, $\mathsf{event}\ e_j(M_{j0}',\dots,M_{jm_j}'); P_j$ is a subprocess of Q_0, so P_j follows F_j. By Proposition 7, $E_j, \sigma_j, \mathcal{E}_j \vdash \mathcal{F}_{P_j}$, where $\mathcal{E}_j$ is the prefix of $\mathcal{E}$ until and including the considered occurrence of the event $e_j(a_{j0},\dots,a_{jm_j})$. By Proposition 6, E is an extension of E_j, so $E, \sigma_j, \mathcal{E} \vdash \mathcal{F}_{P_j}$. Let θ_j' be the substitution that renames replication indices at P_j to fresh replication indices, such that $\mathcal{F}_{F_j,P_j} = \theta_j'\mathcal{F}_{P_j} \cup \{\theta_j' M_{jk}' = M_{jk} \mid k \leq m_j\}$ and $I_{P_j} = \theta_j' M_{j0}'$ since the tuple of replication indices at P_j is added as first argument M_{j0}' of events in Q_0. Let σ_j' be such that $\sigma_j = \sigma_j'\theta_j'$. Then $E, \sigma_j', \mathcal{E} \vdash \theta_j \mathcal{F}_{P_j}$. For all $k \leq m_j$, since $E_j, \sigma_j, M_{jk}' \Downarrow a_{jk}$, we have $E, \sigma_j', \theta_j' M_{jk}' \Downarrow a_{jk}$. We have $\rho, M_{jk} \Downarrow a_{jk}$. Hence $E, \sigma_j' \oplus \rho, \mathcal{E} \vdash \theta_j' M_{jk}' = M_{jk}$, where $\sigma_j' \oplus \rho$ denotes the function that maps x to $\sigma_j'(x)$ when $x \in \mathrm{Dom}(\sigma_j')$ and i to $\rho(i)$ when $i \in \mathrm{Dom}(\rho)$. So $E, \sigma_j' \oplus \rho, \mathcal{E} \vdash \mathcal{F}_{F_j,P_j}$.

Therefore, for each $j \leq m$, there exists a process P_j that follows F_j such that

- for all $j \leq m$, there is a reduction

$$E_j, (\sigma_j, \mathsf{event}\ e_j(M_{j0}',\dots,M_{jm_j}'); P_j), \mathcal{Q}_j, \mathcal{C}_j \xrightarrow{e_j(a_{j0},\dots,a_{jm_j})}_{1,Ev} E_j, (\sigma_j, P_j), \mathcal{Q}_j, \mathcal{C}_j$$

  in the trace $\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$, and if $F_j = \mathsf{inj\text{-}event}(e_j(\dots))$, then $\tau_j \neq \bot$ and $\mathcal{E}(\tau_j) = e_j(a_{j0},\dots,a_{jm_j})$;

- letting $\rho' = \sigma_1' \oplus \dots \oplus \sigma_m' \oplus \rho$, we have $\mathrm{Dom}(\rho') = \mathrm{var}(I_{P_1}) \cup \dots \cup \mathrm{var}(I_{P_m}) \cup \mathrm{var}(\psi)$, $E, \rho', \mathcal{E} \vdash \mathcal{F}_{F_1,P_1} \cup \dots \cup \mathcal{F}_{F_m,P_m}$ and for all $j \leq m$, $\rho'(I_{P_j}) = \sigma_j'(I_{P_j}) = \sigma_j(M_{j0}') = a_{j0}$.

Let $\theta = \theta(P_1,\dots,P_m)$. Let $\mathcal{F} = \mathcal{F}_{F_1,P_1} \cup \dots \cup \mathcal{F}_{F_m,P_m}$, $\mathcal{I} = \{j \mapsto I_{P_j} \mid F_j \text{ is an injective event}\}$, and $\mathcal{V} = \mathrm{var}(I_{P_1}) \cup \dots \cup \mathrm{var}(I_{P_m}) \cup \mathrm{var}(\psi)$. Since $E, \rho', \mathcal{E} \vdash \mathcal{F}_{F_1,P_1} \cup \dots \cup \mathcal{F}_{F_m,P_m}$ and $\neg E, \rho', \mathcal{E} \vdash \mathcal{F}(P_1,\dots,P_m)$, we have $E, \rho', \mathcal{E} \vdash \mathrm{formula}(\mathcal{F} \models\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)$. We extend ρ to all $x \in \mathrm{var}(\phi) \setminus \mathrm{var}(\psi)$ in such a way that $E, \rho', \theta(x) \Downarrow \rho(x)$. Then, for all $x \in \mathrm{var}(\phi) \cup \mathrm{var}(\psi)$, $E, \rho', \theta(x) \Downarrow \rho(x)$. By Lemma 5, there exists ϕ^τ such that $\rho, \mathcal{E} \vdash^{\phi^\tau} \phi$ and, if τ is a non-bottom leaf of ϕ^τ and $\mathcal{S}$ the corresponding leaf of $\mathcal{C}$, then $E, \rho', \mathcal{E} \vdash \mathcal{E}(\tau) = \mathsf{event}(e(M_0'',\dots,M_m''))$ for some $\mathsf{event}(e(M_0'',\dots,M_m'')) \in \mathcal{F}$ and $(\mathcal{F}, M_0'', \mathcal{I}, \mathcal{V}) \in \mathcal{S}$.

We define $\mathbb{F}$ as the function that maps ψ^τ to ϕ^τ build as above. It suffices to show that $\mathbb{F}$ is component-wise injective. Let f be a projection of $\mathbb{F}$ to a leaf of ϕ, and $\mathcal{S}$ the corresponding leaf of $\mathcal{C}$. Assume that $f(\psi_1^\tau) = f(\psi_2^\tau) = \tau \neq \bot$. Let us show that $\psi_1^\tau = \psi_2^\tau$.

Assume that $\psi_1^\tau = \tau_{11} \wedge \dots \wedge \tau_{1m}$ and $\psi_2^\tau = \tau_{21} \wedge \dots \wedge \tau_{2m}$. By construction of $\mathbb{F}$, we have

- for all $j \leq m$, there is a reduction

$$E_{1j}, (\sigma_{1j}, \mathsf{event}\ e_j(M_{1j0}',\dots,M_{1jm_j}'); P_{1j}), \mathcal{Q}_{1j}, \mathcal{C}_{1j} \xrightarrow{e_j(a_{1j0},\dots,a_{1jm_j})}_{1,Ev} E_{1j}, (\sigma_{1j}, P_{1j}), \mathcal{Q}_{1j}, \mathcal{C}_{1j}$$

  in the trace $\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$, and if $F_j = \mathsf{inj\text{-}event}(e_j(\dots))$, then $\tau_{1j} \neq \bot$

and $\mathcal{E}(\tau_{1j}) = e_j(a_{1j0},\dots,a_{1jm_j})$; $E,\rho'_1,\mathcal{E} \vdash \mathcal{F}_1$, for all $j \leq m$, $\rho'_1(I_{P_{1j}}) = a_{1j0}$, $\mathcal{I}_1 = \{j \mapsto I_{P_{1j}} \mid F_j \text{ is an injective event}\}$, $E,\rho'_1,\mathcal{E} \vdash \mathcal{E}(\tau) = \mathsf{event}(e(M''_1,\dots))$, $(\mathcal{F}_1, M''_1, \mathcal{I}_1, \mathcal{V}_1) \in \mathcal{S}$, and $\mathrm{Dom}(\rho'_1) = \mathcal{V}_1$;

- for all $j \leq m$, there is a reduction

  $$E_{2j}, (\sigma_{2j}, \mathsf{event}\ e_j(M'_{2j0},\dots,M'_{2jm_j}); P_{2j}), \mathcal{Q}_{2j}, \mathcal{C}_{2j} \xrightarrow{e_j(a_{2j0},\dots,a_{2jm_j})}_{1,Ev} E_{2j}, (\sigma_{2j}, P_{2j}), \mathcal{Q}_{2j}, \mathcal{C}_{2j}$$

  in the trace $\mathrm{initConfig}(C[Q_0]) \xrightarrow{\mathcal{E}}_{p,\mathcal{T}} E,(\sigma,P),\mathcal{Q},\mathcal{C}$, and if $F_j = \mathsf{inj\text{-}event}(e_j(\dots))$, then $\tau_{2j} \neq \bot$ and $\mathcal{E}(\tau_{2j}) = e_j(a_{2j0},\dots,a_{2jm_j})$; $E,\rho'_2,\mathcal{E} \vdash \mathcal{F}_2$, for all $j \leq m$, $\rho'_2(I_{P_{2j}}) = a_{2j0}$, $\mathcal{I}_2 = \{j \mapsto I_{P_{2j}} \mid F_j \text{ is an injective event}\}$, $E,\rho'_2,\mathcal{E} \vdash \mathcal{E}(\tau) = \mathsf{event}(e(M''_2,\dots))$, $(\mathcal{F}_2, M''_2, \mathcal{I}_2, \mathcal{V}_2) \in \mathcal{S}$, and $\mathrm{Dom}(\rho'_2) = \mathcal{V}_2$.

Let θ'' be a renaming that maps variables of $\mathcal{V}_2$ to distinct fresh variables. Let ρ' be defined by $\rho'(x) = \rho'_1(x)$ if $x \in \mathcal{V}_1$ and $\rho'(x) = \rho'_2(\theta''^{-1}(x))$ if $x \in \theta''(\mathcal{V}_2)$.

Then $E,\rho',\mathcal{E} \vdash \mathcal{F}_1$, $E,\rho',\mathcal{E} \vdash \theta''\mathcal{F}_2$, $E,\rho',\mathcal{E} \vdash \mathsf{event}(e(M''_1,\dots)) = \mathcal{E}(\tau) = \theta''\mathsf{event}(e(M''_2,\dots))$, so $E,\rho',\mathcal{E} \vdash M''_1 = \theta'' M''_2$. Hence by hypothesis, $E,\rho',\mathcal{E} \vdash \bigwedge_{j\in\mathrm{Dom}(\mathcal{I}_1)} \mathcal{I}_1(j) = \theta''\mathcal{I}_2(j)$, so for all $j \in \mathrm{Dom}(\mathcal{I}_1)$, $\rho'(\mathcal{I}_1(j)) = \rho'(\theta''\mathcal{I}_2(j))$, that is, $\rho'_1(I_{P_{1j}}) = \rho'_2(I_{P_{2j}})$, so $a_{1j0} = a_{2j0}$. By Lemma 6, for all $j \in \mathrm{Dom}(\mathcal{I}_1)$, that is, for all j such that F_j is an injective event, there is a single reduction in the trace with a label of the form $e_j(a_{1j0},\dots)$, so $\tau_{1j} = \tau_{2j}$. Furthermore, for all j such that F_j is a non-injective event, $\tau_{1j} = \tau_{2j} = \bot$. So $\psi_1^\tau = \psi_2^\tau$.

Hence $\mathbb{F}$ is component-wise injective, so $\mathcal{E} \vdash \psi \Rightarrow \phi$. This concludes the proof. □

C.3. Authenticated Key Exchange

Proof of Proposition 4 We first show that Q_0 is a secure mutual authentication protocol. The first condition of Definition 7 holds by hypothesis, and it implies the first condition of Definition 6. The last two conditions of Definition 6 come from (5) and (6), as in Proposition 3.

Next, we show the second condition of Definition 7. We define a process Q_1 obtained from Q_0 by adding $\mathsf{event}\ \mathit{part}_A(Y, \mathrm{sid}'(x_1,\dots,x_{r-1})); \mathsf{event}\ \mathit{full}_A(Y, k_A, \mathrm{sid}(x_1,\dots,x_r))$ just before $\overline{c_{Ar}[i_A]}\langle x_r, \mathrm{accept}_A(Y)\rangle$, $\mathsf{event}\ \mathit{part}_B(X, \mathrm{sid}'(y_1,\dots,y_{r-1}))$ just before $\overline{c_{Br-1}[i_B]}\langle y_{r-1}\rangle$, and $\mathsf{event}\ \mathit{full}_B(X, k_B, \mathrm{sid}(y_1,\dots,y_r))$ just before $\overline{c_{Br+1}[i_B]}\langle \mathrm{accept}_B(X)\rangle$. Let Q_2 be the process obtained from Q'_0 by deleting events.

We define

$$Q_{k'_A} = !^{i\leq n'} c(u_S : [1,n]); \mathsf{if\ defined}(k'_A[u_S])\ \mathsf{then}\ \overline{c}\langle k'_A[u_S]\rangle$$

$$Q'_{k'_A} = !^{i\leq n'} c(u_S : [1,n]); \mathsf{if\ defined}(k'_A[u_S])\ \mathsf{then}\ \mathsf{find}\ u' \leq n'\ \mathsf{suchthat\ defined}(y[u'], u_S[u']) \wedge u_S[u'] = u_S\ \mathsf{then}\ \overline{c}\langle y[u']\rangle\ \mathsf{else\ new}\ y : T; \overline{c}\langle y\rangle$$

Since Q'_0 preserves the secrecy of k'_A, we have $Q_2 \mid Q_{k'_A} \approx Q_2 \mid Q'_{k'_A}$.

Below, we define a process Q_{ST} that simulates the test queries of Q_T by calling the process $Q'_0 \mid Q_{k'_A}$ and the test queries of Q'_T by calling $Q'_0 \mid Q'_{k'_A}$, so that

$$Q_1 \mid Q_T \approx \mathsf{newChannel}\ \widetilde{c'}; ((Q'_0 \mid Q_{k'_A})\{\widetilde{c'}/\widetilde{c}\} \mid Q_{ST})$$
$$Q_1 \mid Q'_T \approx \mathsf{newChannel}\ \widetilde{c'}; ((Q'_0 \mid Q'_{k'_A})\{\widetilde{c'}/\widetilde{c}\} \mid Q_{ST})$$

where $\widetilde{c} = (c_{A0},\dots,c_{Ar},c_{AK},c_{B1},\dots,c_{Br},c_{BK},c)$ and $\widetilde{c'}$ consists of fresh names such that $\widetilde{c'} = (c'_{A0},\dots,c'_{Ar}, c'_{AK},c'_{B1},\dots,c'_{Br},c'_{BK},c')$. By deleting events, we have

$$Q_0 \mid Q_T \approx \mathsf{newChannel}\ \widetilde{c'}; ((Q_2 \mid Q_{k'_A})\{\widetilde{c'}/\widetilde{c}\} \mid Q_{ST})$$
$$Q_0 \mid Q'_T \approx \mathsf{newChannel}\ \widetilde{c'}; ((Q_2 \mid Q'_{k'_A})\{\widetilde{c'}/\widetilde{c}\} \mid Q_{ST})$$

Since $Q_2 \mid Q_{k'_A} \approx Q_2 \mid Q'_{k'_A}$, we have by renaming $(Q_2 \mid Q_{k'_A})\{\widetilde{c'}/\widetilde{c}\} \approx (Q_2 \mid Q'_{k'_A})\{\widetilde{c'}/\widetilde{c}\}$. Moreover, Q_{ST} does not use the variables of $Q_2, Q_{k'_A}, Q'_{k'_A}$, so by Lemma 2, Property 2, $\mathsf{newChannel}\ \widetilde{c'}; ((Q_2 \mid Q_{k'_A})\{\widetilde{c'}/\widetilde{c}\} \mid Q_{ST}) \approx \mathsf{newChannel}\ \widetilde{c'}; ((Q_2 \mid Q'_{k'_A})\{\widetilde{c'}/\widetilde{c}\} \mid Q_{ST})$.

Then $Q_0 \mid Q_T \approx \mathsf{newChannel}\ \widetilde{c'}; ((Q_2 \mid Q_{k'_A})\{\widetilde{c'}/\widetilde{c}\} \mid Q_{ST}) \approx \mathsf{newChannel}\ \widetilde{c'}; ((Q_2 \mid Q'_{k'_A})\{\widetilde{c'}/\widetilde{c}\} \mid Q_{ST}) \approx Q_0 \mid Q'_T$, so by transitivity $Q_0 \mid Q_T \approx Q_0 \mid Q'_T$, which proves the desired result.

We now define the process Q_{ST}; we explain this definition below. We define $\widetilde{x}[M]$ as an abbreviation for $x_1[M],\dots,x_r[M]$ and we define $\widetilde{x'}[M]$, $\widetilde{y}[M]$, and $\widetilde{y'}[M]$ similarly. We let $Q_{ST} = Q_{STA} \mid Q_{STB} \mid Q_{RA} \mid Q_{RB}$ where

$$\begin{aligned}
&Q_{STA} = !^{i\leq n_T} \mathit{test}_A[i](u_A);\\
&\quad \mathsf{if\ defined}(x'_{r+1}[u_A]) \wedge x'_{r+1}[u_A] \neq \mathrm{reject}\ \mathsf{then}\\
&\quad \mathsf{if}\ x'_{r+1}[u_A] \neq \mathrm{accept}_A(B)\ \mathsf{then}\\
&\qquad \overline{c'_{AK}[u_A]}\langle\rangle; c'_{AK}[u_A](k); \overline{\mathit{test}_A[i]}\langle k\rangle\\
&\quad \mathsf{else}\\
&\quad \mathsf{find}\ u \leq n_T\ \mathsf{suchthat\ defined}(u_A[u], r_A[u]) \wedge\\
&\qquad u_A[u] = u_A\ \mathsf{then}\ \overline{\mathit{test}_A[i]}\langle r_A[u]\rangle\ \mathsf{else}\\
&\quad \mathsf{find}\ u \leq n_T\ \mathsf{suchthat\ defined}(u_B[u], r_B[u],\\
&\qquad \widetilde{x'}[u_A], \widetilde{y'}[u_B[u]]) \wedge \mathrm{sid}(\widetilde{x'}[u_A]) = \mathrm{sid}(\widetilde{y'}[u_B[u]])\\
&\qquad \mathsf{then}\ \overline{\mathit{test}_A[i]}\langle r_B[u]\rangle\ \mathsf{else}\\
&\quad \overline{c'[i]}\langle u_A\rangle; c'[i](r_A); \overline{\mathit{test}_A[i]}\langle r_A\rangle
\end{aligned}$$

$Q_{STB} = !^{i \leq n_T} test_B[i](u_B);$
$\quad$ if $\mathsf{defined}(y'_{r+1}[u_B]) \wedge y'_{r+1}[u_B] \neq \mathrm{reject}$ then
$\quad$ if $y'_{r+1}[u_B] \neq \mathrm{accept}_B(A)$ then
$\quad\quad \overline{c'_{BK}[u_B]}\langle\rangle; c'_{BK}[u_B](k); \overline{test_B[i]}\langle k\rangle$
$\quad$ else
$\quad$ find $u \leq n_T$ suchthat $\mathsf{defined}(u_B[u], r_B[u]) \wedge$
$\quad\quad u_B[u] = u_B$ then $\overline{test_B[i]}\langle r_B[u]\rangle$ else
$\quad$ find $u \leq n_T$ suchthat $\mathsf{defined}(u_A[u], r_A[u],$
$\quad\quad \widetilde{x}'[u_A[u]], \widetilde{y}'[u_B]) \wedge \mathrm{sid}(\widetilde{x}'[u_A[u]]) = \mathrm{sid}(\widetilde{y}'[u_B])$
$\quad\quad$ then $\overline{test_B[i]}\langle r_A[u]\rangle$ else
$\quad$ find $u'_A \leq n$ suchthat $\mathsf{defined}(\widetilde{x}'[u'_A], \widetilde{y}'[u_B],$
$\quad\quad x'_{r+1}[u'_A]) \wedge \mathrm{sid}(\widetilde{x}'[u'_A]) = \mathrm{sid}(\widetilde{y}'[u_B]) \wedge$
$\quad\quad x'_{r+1}[u'_A] = \mathrm{accept}_A(B)$ then
$\quad\quad \overline{c'[i+n_T]}\langle u'_A\rangle; c'[i+n_T](r_B); \overline{test_B[i]}\langle r_B\rangle$
$Q_{RA} = !^{i \leq n} c_{A0}[i](Y'); \overline{c'_{A0}[i]}\langle Y'\rangle;$
$\quad c'_{A1}[i](x'_1); \overline{c_{A1}[i]}\langle x'_1\rangle; c_{A2}[i](x'_2); \overline{c'_{A2}[i]}\langle x'_2\rangle;$
$\quad \ldots; c'_{Ar}[i](x'_r, x'_{r+1}); \overline{c_{Ar}[i]}\langle x'_r, x'_{r+1}\rangle$
$Q_{RB} = !^{i \leq n} c_{B1}[i](y'_1); \overline{c'_{B1}[i]}\langle y'_1\rangle;$
$\quad c'_{B2}[i](y'_2); \overline{c_{B2}[i]}\langle y'_2\rangle; \ldots; c_{Br}[i](y'_r); \overline{c'_{Br}[i]}\langle y'_r\rangle;$
$\quad c'_{Br+1}[i](y'_{r+1}); \overline{c_{Br+1}[i]}\langle y'_{r+1}\rangle$

where $I_\eta(n') = 2 \times I_\eta(n_T)$ (n' is the number of queries allowed in $Q_{k'_A}$ and $Q'_{k'_A}$) and all variables in these processes are fresh. (The variables Y', x'_j, y'_j play the same role as Y, x_j, y_j in Q'_0; they have been renamed to avoid confusion with the variables of Q'_0.) The processes Q_{RA} and Q_{RB} relay the requests from channels c_{Aj} and c_{Bj} to channels c'_{Aj} and c'_{Bj}. These relay processes are useful in order to store the messages in $x'_1, \ldots, x'_{r+1}, y'_1, \ldots, y'_{r+1}$, to have access to them without reading the variables of Q'_0.

The processes Q_{STA} and $_{STB}$ simulate the test queries. They first check that the queried copy of Q_A or Q_B has accepted (first test of Q_{STA} and Q_{STB}). Then, if the queried session is not between A and B, they call Q'_0 to return the session key. Otherwise, they call $Q_{k'_A}$ to return the key of sessions between A and B, or $Q'_{k'_A}$ to return a fresh random number for each session between A and B. Before calling $Q_{k'_A}$ or $Q'_{k'_A}$, they first check if the same test query (or a test query to the partner) has already been called, and if it has, they return the previously returned value. (These checks are not strictly necessary, because $Q'_{k'_A}$ already checks if the same query has already been called. However, they slightly simplify the proof by making the structure of Q_{STA} and Q_{STB} closer to the structure Q'_{TA} and Q'_{TB}.) After these checks, Q_{STA} calls $Q_{k'_A}$ or $Q'_{k'_A}$ directly (last line of Q_{STA}) while Q_{STB} first uses a find to find the copy of Q_A, partner of the considered session and calls $Q_{k'_A}$ or $Q'_{k'_A}$ for that partner (last find of Q_{STB}).

To show an equivalence $L \approx R$, we show that, after excluding a set of traces of negligible probability, each trace of L can be simulated by a trace of R of the same probability, and conversely.

For the equivalence

$$Q_1 \mid Q_T \approx \mathsf{newChannel}\ \widetilde{c}'; ((Q'_0 \mid Q_{k'_A})\{\widetilde{c}'/\widetilde{c}\} \mid Q_{ST})$$

the proof is done by considering only the traces in which the correspondences (5)–(7) hold. The other traces have negligible probability since the correspondences (5)–(7) are satisfied by Q'_0 with public variables $\{k'_A\}$, so by Lemma 1, they are also satisfied by $\mathsf{newChannel}\ \widetilde{c}'; ((Q'_0 \mid Q_{k'_A})\{\widetilde{c}'/\widetilde{c}\} \mid Q_{ST})$ and $\mathsf{newChannel}\ \widetilde{c}'; ((Q'_0 \mid Q'_{k'_A})\{\widetilde{c}'/\widetilde{c}\} \mid Q_{ST})$. We establish the correspondence between traces by induction on the length of the trace:

- When $Q_1 \mid Q_T$ receives a message on channel $c_{Aj}[i_A]$ with $j < r-1$, Q_1 stores the received message in $x_j[i_A]$, answers by returning the next message of the protocol $x_{j+1}[i_A]$ on $c_{Aj+1}[i_A]$. Correspondingly, in $((Q'_0 \mid Q_{k'_A})\{\widetilde{c}'/\widetilde{c}\} \mid Q_{ST})$, Q_{RA} stores the received message in $x'_j[i_A]$ (or $Y'[i_A]$ if $j = 0$), forwards it on $c'_{Aj}[i_A]$; $Q'_0\{\widetilde{c}'/\widetilde{c}\}$ answers to it like Q_1 except that the next message $x_{j+1}[i_A]$ is sent on $c'_{Aj+1}[i_A]$; Q_{RA} then stores this message in $x'_{j+1}[i_A]$ and forwards it on $c_{Aj+1}[i_A]$. When $j = r-1$, the situation is similar, except that the returned message is a pair $x_r[i_A], \mathrm{accept}_A(Y[i_A])$ or $x_r[i_A], \mathrm{reject}$, stored by Q_{RA} in $x'_r[i_A], x'_{r+1}[i_A]$. When $j = r-1$ and the protocol accepts, both sides define $k_A[i_A]$, execute event $full_A(Y[i_A], k_A[i_A], \mathrm{sid}(\widetilde{x}[i_A]))$, and send $x_r[i_A], \mathrm{accept}_A(Y[i_A])$, so in the right-hand side $x'_{r+1}[i_A] = \mathrm{accept}_A(Y[i_A])$. When furthermore $Y[i_A] = B$, Q'_0 additionally defines $k'_A[i_A] = k_A[i_A]$. When $j = r-1$ and the protocol rejects, $k_A[i_A]$ is not defined and both sides send $x_r[i_A], \mathrm{reject}$, so in the right-hand side $x'_{r+1}[i_A] = \mathrm{reject}$.

  So, $k_A[i_A]$ is defined in the left-hand side if and only if $x'_{r+1}[i_A]$ is defined and different from reject in the right-hand side, and in this case, $Y[i_A]$, $\widetilde{x}[i_A]$, and $k_A[i_A]$ have the same value in both sides of the equivalence and, in the right-hand side, $Y'[i_A] = Y[i_A]$, $\widetilde{x}'[i_A] = \widetilde{x}[i_A]$, $k'_A[i_A] = k_A[i_A]$ if $Y[i_A] = B$, $k'_A[i_A]$ is not defined if $Y[i_A] \neq B$, and $x'_{r+1}[i_A] = \mathrm{accept}_A(Y[i_A])$.

- Similarly, the messages on $c_{Bj}[i_B]$ are answered in the same way by both sides of the equivalence, thanks to the forwarding by Q_{RB} in the right-hand side.

So, $k_B[i_B]$ is defined in the left-hand side if and only if $y'_{r+1}[i_B]$ is defined and different from reject in the right-hand side, and in this case, $X[i_B]$, $\widetilde{y}[i_B]$, and $k_B[i_B]$ have the same value in both sides of the equivalence and, in the right-hand side, $X'[i_B] = X[i_B]$, $\widetilde{y'}[i_B] = \widetilde{y}[i_B]$, and $y'_{r+1}[i_B] = \mathrm{accept}_B(X[i_B])$.

- When $Q_1 \mid Q_T$ receives a message $test_A[i](u_A)$, Q_{TA} returns $k_A[u_A]$ if it is defined. Correspondingly, in the right-hand side, Q_{STA} first tests if $x'_{r+1}[u_A]$ is defined and different from reject, which is equivalent to $k_A[u_A]$ defined, as mentioned above.

  If $x'_{r+1}[u_A] \neq \mathrm{accept}_A(B)$, then $Y[u_A] \neq B$. In this case, Q_{STA} sends an empty message on $c'_{AK}[u_A]$. Q'_0 receives it, and sends $k_A[u_A]$ on $c'_{AK}[u_A]$. Q_{STA} then receives this message, stores it in k, and sends $k = k_A[u_A]$ on $test_A[i]$, as in the left-hand side.

  Otherwise, $x'_{r+1}[u_A] = \mathrm{accept}_A(B)$ and $Y[u_A] = B$. Then Q_{STA} checks if the same test query has been asked before (test query number u such that $u_A[u]$ and $r_A[u]$ are defined, and $u_A[u] = u_A$). Below, we show that, when $r_A[i]$ is defined, $k_A[u_A[i]]$ and $Y[u_A[i]]$ are defined, $r_A[i] = k_A[u_A[i]]$, and $Y[u_A[i]] = B$. So $r_A[u] = k_A[u_A[u]] = k_A[u_A]$, and $k_A[u_A]$ is sent on $test_A[i]$, as in the left-hand side.

  Next, Q_{STA} checks if a test query has been asked to the partner of $Q_A^{u_A}$ (test query number u such that $u_B[u]$, $r_B[u]$, $\widetilde{x'}[u_A]$, and $\widetilde{y'}[u_B[u]]$ are defined and $\mathrm{sid}(\widetilde{x'}[u_A]) = \mathrm{sid}(\widetilde{y'}[u_B[u]])$). Below, we show that, when $r_B[i]$ is defined, $k_B[u_B[i]]$ and $X[u_B[i]]$ are defined, $r_B[i] = k_B[u_B[i]]$, and $X[u_B[i]] = A$. So $r_B[u] = k_B[u_B[u]]$. Since $k_B[u_B[u]]$ and $x'_r[u_A]$ are defined, the events $\mathit{full}_B(X[u_B[u]], k_B[u_B[u]], \mathrm{sid}(\widetilde{y}[u_B[u]]))$ and $\mathit{full}_A(Y[u_A], k_A[u_A], \mathrm{sid}(\widetilde{x}[u_A]))$ have been executed. Since $\mathrm{sid}(\widetilde{x}[u_A]) = \mathrm{sid}(\widetilde{x'}[u_A]) = \mathrm{sid}(\widetilde{y'}[u_B[u]]) = \mathrm{sid}(\widetilde{y}[u_B[u]])$, $X[u_B[u]] = A$, and $Y[u_A] = B$, these events are $\mathit{full}_B(A, k_B[u_B[u]], \mathrm{sid}(\widetilde{x}[u_A]))$ and $\mathit{full}_A(B, k_A[u_A], \mathrm{sid}(\widetilde{x}[u_A]))$. So by the correspondence (7), $k_B[u_B[u]] = k_A[u_A]$, hence $r_B[u] = k_B[u_B[u]] = k_A[u_A]$ is sent on $test_A[i]$, as in the left-hand side.

  Finally, if both finds fail, then Q_{STA} sends u_A on $c'[i]$. $Q_{k'_A}\{\widetilde{c'}/\widetilde{c}\}$ receives this message and replies by sending $k'_A[u_A] = k_A[u_A]$ on $c'[i]$. Q_{STA} stores the reply in r_A, so $r_A = k_A[u_A]$, and sends $k_A[u_A]$ on $test_A[i]$, as in the left-hand side. Moreover, we have $Y[u_A] = B$ so, spelling out all array indices, $r_A[i] = k_A[u_A[i]]$ and $Y[u_A[i]] = B$.

- When $Q_1 \mid Q_T$ receives a message $test_B[i](u_B)$, the situation is almost symmetric of the previous case. We just detail the case in which $y'_{r+1}[u_B] = \mathrm{accept}_B(A)$ and the first two finds of Q_{STB} fail. We have $X[u_B] = A$. Then the event $\mathit{full}_B(A, k_B[u_B], \mathrm{sid}(\widetilde{y}[u_B]))$ has been executed. By the correspondence (6), the event $\mathit{full}_A(B, k_B[u_B], \mathrm{sid}(\widetilde{y}[u_B]))$ has been executed. So there exists u''_A such that $Y'[u''_A] = Y[u''_A] = B$, $k_A[u''_A] = k_B[u_B]$, $\mathrm{sid}(\widetilde{x'}[u''_A]) = \mathrm{sid}(\widetilde{x}[u''_A]) = \mathrm{sid}(\widetilde{y}[u_B])$, $x'_{r+1}[u''_A] = \mathrm{accept}_A(B)$. So the last find of Q_{STB} succeeds for some value of u'_A. Moreover, since $x'_r[u'_A]$ is defined, the event $\mathit{full}_A(Y[u'_A], k_A[u'_A], \mathrm{sid}(\widetilde{x}[u'_A]))$ has been executed. Since $x'_{r+1}[u'_A] = \mathrm{accept}_A(B)$, $Y[u'_A] = Y'[u'_A] = B$ and $\mathrm{sid}(\widetilde{x}[u'_A]) = \mathrm{sid}(\widetilde{x'}[u'_A]) = \mathrm{sid}(\widetilde{y'}[u_B]) = \mathrm{sid}(\widetilde{y}[u_B])$, this event is $\mathit{full}_A(B, k_A[u'_A], \mathrm{sid}(\widetilde{y}[u_B]))$. By the correspondence (7), $k_A[u'_A] = k_B[u_B]$. The process Q_{STB} sends u'_A on channel $c'[i + n_T]$. This message is received by $Q_{k'_A}$. Moreover, $k'_A[u'_A]$ is defined and $k'_A[u'_A] = k_A[u'_A]$, since $x'_r[u'_A]$ is defined and $Y[u'_A] = B$. Then $Q_{k'_A}$ replies by sending $k'_A[u'_A]$ on channel $c'[i + n_T]$. Then $r_B = k'_A[u'_A] = k_A[u'_A] = k_B[u_B]$, and $k_B[u_B]$ is sent on $test_B[i]$, as in the left-hand side.

For the equivalence

$$Q_1 \mid Q'_T \approx \mathsf{newChannel}\ \widetilde{c'}; ((Q'_0 \mid Q'_{k'_A})\{\widetilde{c'}/\widetilde{c}\} \mid Q_{ST})$$

we exclude not only the traces that do not satisfy the correspondences (5)–(7), but also the traces in which $k'_A[u] = k'_A[u']$ for some $u \neq u'$. These traces have negligible probability, because otherwise that would contradict the secrecy of k'_A: the adversary could distinguish $Q'_0 \mid Q_{k'_A}$ from $Q'_0 \mid Q'_{k'_A}$ with non-negligible probability, by detecting the former when he obtains the same answer to queries $\overline{c'[i]}\langle u\rangle$ and $\overline{c'[i]}\langle u'\rangle$ for some $u \neq u'$. For this equivalence, the cases of protocol messages are similar to the previous equivalence, so we only detail the cases of test queries.

- When $Q_1 \mid Q'_T$ receives a message $test_A[i](u_A)$, Q'_{TA} first tests if $k_A[u_A]$ and $Y[u_A]$ are defined. Correspondingly, in the right-hand side, Q_{STA} first tests if $x'_{r+1}[u_A]$ is defined and different from reject, which is equivalent to $k_A[u_A]$ and $Y[u_A]$ defined.

  Next, if $Y[u_A] \neq B$, then $Q_1 \mid Q'_T$ sends $k_A[u_A]$ on $test_A[i]$. Correspondingly, in the right hand-side, if $y'_{r+1}[u_A] \neq \mathrm{accept}_A(B)$, that is, $Y[u_A] \neq B$, then Q_{STA} sends a message on $c'_{AK}[u_A]$. Q'_0 receives it, and replies by sending $k_A[u_A]$ on $c'_{AK}[u_A]$. Q_{STA} receives this message, and sends $k = k_A[u_A]$ on $test_A[i]$, as in the left-hand side.

  Otherwise, both sides execute two finds that yield the same result because $\widetilde{x}[u_A] = \widetilde{x'}[u_A]$, $\widetilde{y}[u_B] = \widetilde{y'}[u_B]$, and as we shall see below $r_A[u]$ and $r_B[u]$ have the same value in both sides of the equivalence.

Finally, when both finds fail, in the left-hand side, Q'_{TA} sends a fresh random number uniformly distributed in $I_\eta(T)$ on $test_A[i]$. Correspondingly, in the right-hand side, Q_{STA} sends u_A on $c'[i]$. $Q'_{k'_A}$ receives this message. It checks that $k'_A[u_A]$ is defined, which is true because $k_A[u_A]$ is defined and $Y[u_A] = B$. Next, it looks for a previous query with the same u_A; there is no such query, because otherwise one of the previous finds would have succeeded:

  - If u_A was previously sent on $c'[i']$ by Q_{STA}, then there would be an u ($u = i'$) such that $u_A[u]$ and $r_A[u]$ are defined and $u_A[u] = u_A$, so the first find would have succeeded.
  - If u_A was previously sent on $c'[i'+n_T]$ by Q_{STB}, then there would be an u ($u = i'$) such that $u'_A[u] = u_A$, $\mathrm{sid}(\widetilde{x}'[u'_A[u]]) = \mathrm{sid}(\widetilde{y}'[u_B[u]])$, and these values are defined, so the second find would have succeeded.

  So $Q'_{k'_A}$ replies by sending a fresh random number uniformly distributed in $I_\eta(T)$ on $c'[i]$. Q_{STA} receives it, stores it in $r_A[i]$, and sends it on $test_A[i]$, as in the left-hand side.

- When $Q_1 \mid Q_T$ receives a message $test_B[i](u_B)$, the situation is almost symmetric of the previous case. We only detail the case in which $y'_{r+1}[u_B] = \mathrm{accept}_B(A)$ and the first two finds of Q'_{TB} and Q_{STB} fail. In this case, in the left-hand side, Q'_{TB} sends a fresh random number uniformly distributed in $I_\eta(T)$ on $test_B[i]$. In the right-hand side, as in the proof of the previous equivalence, the last find of Q_{STB} succeeds, $\mathrm{sid}(\widetilde{x}'[u'_A]) = \mathrm{sid}(\widetilde{y}'[u_B])$, $x'_{r+1}[u'_A] = accept_A(B)$, $k_A[u'_A]$ is defined, and $Y[u'_A] = B$. So Q_{STB} sends u'_A on $c'[i+n_T]$. $Q'_{k'_A}$ receives this message. It checks that $k'_A[u'_A]$ is defined, which is true because $k_A[u'_A]$ is defined and $Y[u'_A] = B$. Next, it looks for a previous query with the same u'_A; there is no such query, because otherwise one of the previous finds would have succeeded:

  - If u'_A was previously sent on $c'[i'+n_T]$ by Q_{STB}, then there would be an u ($u = i'$) such that $u'_A[u]$ and $r_B[u]$ are defined and $u'_A[u] = u'_A$. Then $u_B[u]$ is also defined, $\mathrm{sid}(\widetilde{x}'[u'_A[u]]) = \mathrm{sid}(\widetilde{y}'[u_B[u]])$, $x'_{r+1}[u'_A[u]] = accept_A(B)$. So $\mathrm{sid}(\widetilde{y}[u_B[u]]) = \mathrm{sid}(\widetilde{y}'[u_B[u]]) = \mathrm{sid}(\widetilde{x}'[u'_A[u]]) = \mathrm{sid}(\widetilde{x}'[u'_A]) = \mathrm{sid}(\widetilde{y}'[u_B]) = \mathrm{sid}(\widetilde{y}[u_B])$. In order to obtain a contradiction, assume that $u_B[u] \neq u_B$. The event $full_B(A, k_B[u_B[u]], \mathrm{sid}(\widetilde{y}[u_B]))$ has been executed in copy number $u_B[u]$ of Q_B and $full_B(A, k_B[u_B], \mathrm{sid}(\widetilde{y}[u_B]))$ has been executed in copy number u_B of Q_B. Since the correspondence (6) is injective, two distinct events $full_A(B, k_B[u_B[u]], \mathrm{sid}(\widetilde{y}[u_B]))$ and $full_A(B, k_B[u_B], \mathrm{sid}(\widetilde{y}[u_B]))$ have been executed. So $k_B[u_B[u]] = k_A[u_{A1}]$ and $k_B[u_B] = k_A[u_{A2}]$ with $u_{A1} \neq u_{A2}$. Moreover, by the correspondence (7), since the events $full_B(A, k_B[u_B[u]], \mathrm{sid}(\widetilde{y}[u_B]))$ and $full_A(B, k_B[u_B], \mathrm{sid}(\widetilde{y}[u_B]))$ have been executed, $k_B[u_B[u]] = k_B[u_B]$, so $k_A[u_{A1}] = k_A[u_{A2}]$ with $u_{A1} \neq u_{A2}$. This contradicts the exclusion of traces with $k_A[u] = k_A[u']$ for some $u \neq u'$. So $u_B[u] = u_B$.[1] So the first find would have succeeded.
  - If u'_A was previously sent on $c'[i']$ by Q_{STA}, then there would be an u ($u = i'$) such that $u_A[u]$ and $r_A[u]$ are defined and $u_A[u] = u'_A$. Since the last find of Q_{STB} succeeds, $\mathrm{sid}(\widetilde{x}'[u'_A]) = \mathrm{sid}(\widetilde{y}'[u_B])$, so $\mathrm{sid}(\widetilde{x}'[u_A[u]]) = \mathrm{sid}(\widetilde{y}'[u_B])$, so the second find would have succeeded.

  So $Q'_{k'_A}$ replies by sending a fresh random number uniformly distributed in $I_\eta(T)$ on $c'[i + n_T]$. Q_{STB} receives it, stores it in $r_B[i]$, and sends it on $test_B[i]$, as in the left-hand side. $\square$

D. Discussion on Authentication and Key Exchange

We discuss here some choices made in our modeling of authentication and key exchange.

- We have assumed that A plays only the role of the initiator and B plays only the role of the responder. We could also model a situation in which A and B play both roles, by including a process Q'_A for A playing the responder role and a process Q'_B for B playing the initiator role. Which model is more appropriate depends on the protocol and its intended usage: the former model is appropriate for protocols that use distinct keys for the initiator and responder roles, such as SSH for instance.
- We could also extend the framework to protocols that use a trusted server, by including it into Q_S.
- For simplicity, we have assumed that the participants terminate immediately after accepting; we could obviously extend the framework to allow them to accept before the end of the protocol.

[1] More generally, if $\mathrm{sid}(\widetilde{y}[u'_B]) = \mathrm{sid}(\widetilde{y}[u_B])$, then $u'_B = u_B$. So two sessions can have the same session identifiers only with negligible probability.

- [16] uses the notion of matching conversations instead of sessions identifiers. Matching conversations correspond to session identifiers when $\mathrm{sid}(x_1, \ldots, x_r) = (x_1, \ldots, x_r)$ and $\mathrm{sid}'(x_1, \ldots, x_{r-1}) = (x_1, \ldots, x_{r-1})$ with the additional requirement that the messages from A to B are received by B after they are sent by A and symmetrically. We do not consider this requirement here, because it would complicate the verification considerably. We partly compensate for this weaker definition by checking an injective correspondence, while [16] infers injectivity from the correct ordering of messages—see [16, Appendix C]. More recent formalizations [7, 15, 27, 40, 42] use session identifiers as we do.

- It is often required that, with overwhelming probability, distinct sessions have distinct session identifiers. Here, we only require that n sessions of A with the same identifier correspond to n sessions of B with that identifier. For authenticated key exchange, the secrecy of the key combined with the correspondence (7) (which means that two sessions with same identifier have the same key) implies that, with overwhelming probability, distinct sessions have distinct session identifiers.

E. Detailed Experimental Results

In our tests, all protocols are in a configuration in which the honest participants are willing to run sessions with the adversary. Shared-key encryption is implemented as encrypt-then-MAC, where the encryption is IND-CPA (indistinguishability under chosen plaintext attacks) and the MAC is UF-CMA (unforgeability under chosen message attacks); public-key encryption is assumed to be IND-CCA2 (indistinguishability under adaptive chosen ciphertext attacks); signatures are assumed to be UF-CMA.

The session identifier is chosen to contain all messages of the protocol, except messages that are sent to or received from a server (that is, messages that are not between A and B), messages that are just forwarded without checking (those can be changed by the adversary), and signatures when the security definition of signatures allows an adversary to forge a new signature for a message that has already been signed.

For the public key protocols, the prover needs to be given the main proof steps. We detail them below. For shared-key protocols, the proof is fully automatic.

Woo-Lam shared-key [36] This protocol is a one-way authentication protocol, so we prove only the correspondence (4). Our prover cannot prove this correspondence for the original version of the protocol, as there is a known attack against it, but proves it for the corrected version [36].

Woo-Lam public-key [62] The situation is similar to the Woo-Lam shared key protocol. Our prover cannot prove the correspondence (4) for the original version of the protocol, as there is an attack against it, but proves it for the corrected version [64].

In this protocol, the third message is a signature. The proof fails when the signature is included in the session identifier and the security definition of signatures allows an adversary to forge a new signature for a message that has already been signed. Indeed, the signature is not authenticated in this case. The proof succeeds both when the signature is not included in the session identifier and when the security definition of signatures prevents forgeries even for already signed messages.

For both versions of this protocol, we give the following proof steps to prover:

```
SArename Rkey
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 transformation `SArename Rkey` renames the variable `Rkey` to three different names $\mathtt{Rkey}_1$, $\mathtt{Rkey}_2$, and $\mathtt{Rkey}_3$, one for each assignment to `Rkey`, and thus allows us to distinguish these three cases. The instruction `crypto sign rkS` means that the prover should apply the definition of security of signatures (primitive `sign`), for the key generated from random number `rkS`. The instruction `success` means that prover should check whether the desired security properties are proved.

Needham-Schroeder public-key [53] This protocol is a mutual authentication protocol. Our prover shows the correspondence (3) but the proof fails for (4); indeed, there is a well-known attack against it [48]. The prover proves both (3) and (4) for the corrected version [48].

For both versions of this protocol, we give the following proof steps to the prover:

```
SArename Rkey
crypto sign rkS
crypto enc rkA
crypto enc rkB
SArename Nb_29
simplify
SArename Na_21
simplify
success
```

Denning-Sacco public-key [34] This protocol is a key exchange protocol, so we try to prove the hypothesis of Proposition 4. Since there is no message from B to A in this protocol, B is not authenticated to A, so (5) clearly does not hold. (There is in fact no good place for putting the event $part_B$.) For both the original and the corrected version of [5], this protocol is also subject to an obvious replay attack, so unsurprisingly our prover cannot show the injective correspondence (6). Our prover shows (7) for both the original and the corrected version. It shows the secrecy of k'_A and the non-injective correspondence $\mathsf{event}(full_B(A,k,x)) \Rightarrow \mathsf{event}(full_A(B,k,x))$ only for the corrected version. (There is a well-known attack [5] against them in the original version.)

For both versions of this protocol, we give the following proof steps to the prover:

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

The first `success` instruction is useful in order to prove (7): this correspondence is obvious on the initial game, because the key k or k' is computed from the protocol messages contained in the session identifier x. The relation between the key k and the session identifier x is hidden by the subsequent game transformations.

Needham-Schroeder shared-key [53] The proof of secrecy of the key fails for both the original and the corrected version [54]: the protocol contains a key confirmation round $B \to A : \{N_B\}_K$, $A \to B : \{N_B - 1\}_K$ and these messages may reveal information on the key K. However, the prover shows (3) but fails to show (4) for the original version of the protocol. This failure comes from a limitation of our prover: it fails to prove that $N_B[i] \neq N_B[i'] - 1$ with overwhelming probability, where N_B is a nonce. (Proving this property requires distinguishing two cases: when $i = i'$, we have $N_B[i] \neq N_B[i] - 1$; when $i \neq i'$, both sides are independent random numbers, which have a negligible probability of being equal.) The prover shows both (3) and (4) for the corrected version. When the key confirmation round is removed, the prover proves the secrecy of the key k'_A, but fails to prove the authentication (which is indeed wrong).

Yahalom [23] The situation is similar to the Needham-Schroeder shared-key protocol: the proof of secrecy of the key fails because of a key confirmation message $\{N_B\}_K$. The prover still shows (3) and (4). When the key confirmation message is removed, the prover shows (3) but fails to show (4) (which is indeed wrong).

Otway-Rees [55] The prover shows the secrecy of k'_A, but does not show the correspondence properties (5), (6), and (7). These correspondences are indeed wrong: as noticed in [23], each participant may accept while the other participant fails to get the key, so (6) is wrong. The correspondences (5) and (7) are wrong due to replay attacks.