Secure Implementation of Channel Abstractions

Martín Abadi*
abadi@research.bell-labs.com
Bell Labs Research
Lucent Technologies

Cédric Fournet
fournet@microsoft.com
Microsoft Research

Georges Gonthier†
Georges.Gonthier@inria.fr
INRIA Rocquencourt

September 27, 2000

Abstract

Communication in distributed systems often relies on useful abstractions such as channels, remote procedure calls, and remote method invocations. The implementations of these abstractions sometimes provide security properties, in particular through encryption. In this paper we study those security properties, focusing on channel abstractions. We introduce a simple high-level language that includes constructs for creating and using secure channels. The language is a variant of the join-calculus and belongs to the same family as the pi-calculus. We show how to translate the high-level language into a lower-level language that includes cryptographic primitives. In this translation, we map communication on secure channels to encrypted communication on public channels. We obtain a correctness theorem for our translation; this theorem implies that one can reason about programs in the high-level language without mentioning the subtle cryptographic protocols used in their lower-level implementation.

*This work was partly done at Compaq's Systems Research Center.

†Partly supported by ESPRIT CONFER-2 WG-21836.

Contents

1 Communication constructs and security

While cryptography is useful for distributed applications and fun even for application programmers, cryptographic manipulations by and large do not belong in application code. Ideally, application code should not be concerned with the details of key establishment and management, but should instead rely on abstractions and services that encapsulate cryptographic operations. In recent years, several APIs (application program interfaces) for security have appeared, providing such abstractions and services [29, 28, 34, 30, 9, 26, 43]. Although there are substantial differences between these APIs, they generally offer the promise of making application code more modular, simpler, and ultimately more robust.

In this paper, we consider high-level abstractions that largely hide the difficulties of network security from applications. These high-level abstractions support the pleasing illusion that all application address spaces are on the same machine, and that a centralized operating system provides security for them. In reality, these address spaces could be spread across a network, and security could depend on several local operating systems and on cryptographic protocols across machines. Thus, the application code need not be concerned with the security implications of distribution.

For example, when an application requires secure communication between a client and a server, the application code may include a call to create a secure channel [26]. The secure channel is an abstraction, which can be implemented in a variety of ways. In the local case, the implementation can rely on a local operating system, whereas in the distributed case it involves a network connection and a key. Similarly, the application code may include a method invocation, which may look the same whether the method invocation is local or remote [43, 44, 42]. Security for remote method invocations may be guaranteed by the transport implementation, fairly transparently to the application.

Up to now, the design and implementation of abstractions for secure communication has been more an art than a science. In particular, little is known on how to analyze programs that use these abstractions, or on what it means to implement these abstractions correctly. We address these issues, focusing on channel abstractions.

- We introduce a simple high-level language that includes constructs for creating and using secure channels. The language is a variant of the join-calculus [18] and belongs to the same family as the pi-calculus [38, 36, 37]. The properties of secure channels include, for example, protection against eavesdropping and against replays.

- We show how to translate the high-level language into a lower-level language that includes cryptographic primitives (like the spi-calculus [6]).

In this translation, we implement communication on secure channels by encrypted communication on public channels.

- We state and prove a soundness property for our translation. This property guarantees that one can reason about programs in the high-level language independently of their implementation.

- Finally, we consider variants of our translation, bringing it closer to practicality. These variants avoid encryption in cases where it is unnecessary.

Neither the details of our language nor those of our translation should be essential, and hopefully a programme analogue to ours can be carried out for other languages. The corresponding translations might embody implementations with diverse concerns about performance and system structure. However, in our experience, even slight variations in languages or in translations can lead to large complications and sometimes to security flaws.

Our definitions and proofs are built in a modular fashion on top of small cryptographic protocols. In recent years there have been important advances on methods for analyzing cryptographic protocols (e.g., [13]). We rely on some of those advances, but we are not interested in protocols in isolation. We construct only a few protocols and show that their low-level details match the high-level definition of secure channels. To our knowledge, results of this kind have not previously appeared in the literature. It would be interesting (and probably hard) to obtain analogous results for other protocols.

This paper represents the confluence of several projects, including the development of the join-calculus [18, 20, 21, 22, 17] and of secure network objects [43], and the application of the pi-calculus and the spi-calculus for reasoning about security protocols [6, 14]. Those projects encountered some of the questions treated in this paper (see section 2.2.3). They also produced the work most closely related to that described here, including partial encodings of encryption in the pi-calculus [6, 14]. Our translation goes in the direction opposite to those encodings.

In the next section we describe the source language and the target language of our translation, and give an informal overview of some of the themes of the paper. In sections 3 and 4, we present our translation, with several variants, and in sections 5 and 6 we study its theory and some applications of this theory. We conclude in section 7. Appendices contain supplementary formal definitions and proofs of our results.

Parts of this work appear in preliminary form in three conference papers [3, 4, 5]. A fourth conference paper continues this work with a treatment of authentication primitives [5].

2 From the join-calculus to the sjoin-calculus

In this section, we introduce our high-level language and our low-level language. Along the way, we motivate some of the technical ideas developed in later sections. We postpone most formal details to later sections.

2.1 The join-calculus

Our high-level language is the join-calculus. This is a name-passing calculus, like the pi-calculus, with an emphasis on distributed programming. It constitutes the core of a programming language with a fully distributed implementation [22]. The join-calculus also draws on functional languages with pattern matching and on object-oriented languages. More precisely, our starting point is the join-calculus as described in [18]. This join-calculus is as expressive as the asynchronous pi-calculus, and it can be used for writing functional and object-oriented examples rather directly.

2.1.1 Syntax and informal semantics

The syntax of the join-calculus assumes a countable set of names $\mathcal{N}$. We use lowercase identifiers x, y, foo, bar, ... to represent the elements of $\mathcal{N}$.

In the pure join-calculus, as we describe it here, names are used only as names of channels (or ports). Each channel has an associated arity—a fixed, integer size for the tuples passed on the channel. We require that names be used consistently in processes, respecting their arities, and enforce this requirement by adopting a type system. While there exists a rich, polymorphic type system for the join-calculus [21, 17], a simple monomorphic type system suffices for our present purposes. We write $\langle \sigma_1, \ldots, \sigma_n \rangle$ for the type of channels that carry tuples with n components of respective types $\sigma_1, \ldots, \sigma_n$, and restrict attention to types of this form. We allow types to be recursively defined (formally, using a fixpoint operator), so we may have for example $\sigma = \langle \sigma, \sigma \rangle$. We assume that each name is associated with a type (although we usually keep this type implicit), and that there are infinitely many names for each type. Throughout, we consider only well-typed processes.

Furthermore, in the pure join-calculus, the set of values is defined to be the set of names. In extensions, names are included in a larger set of values. In any case, the contents of messages are values, and we use letters u, v, ... to represent values. Further, we write $\widetilde{v}$ for a tuple $v_1, v_2, \ldots, v_n$.

The syntax of the join-calculus includes categories of processes, definitions, and join-patterns. Processes, definitions, and join-patterns are defined recursively in Figure 1. The operator $|$ has highest precedence, so for example $\mathsf{def}\ D\ \mathsf{in}\ P \mid P'$ means $\mathsf{def}\ D\ \mathsf{in}\ (P \mid P')$.

Intuitively, the semantics of processes is as follows.

$$
\begin{array}{lcll}
P & ::= & & \text{processes} \\
 & & x\langle\widetilde{v}\rangle & \quad\text{message} \\
 & | & \mathsf{def}\ D\ \mathsf{in}\ P & \quad\text{local definition} \\
 & | & \mathsf{if}\ v = v'\ \mathsf{then}\ P\ \mathsf{else}\ P' & \quad\text{comparison} \\
 & | & P \mid P' & \quad\text{parallel composition} \\
 & | & \mathsf{0} & \quad\text{null process} \\
 & & & \\
D & ::= & & \text{definitions} \\
 & & J \triangleright P & \quad\text{reaction rule} \\
 & | & D \wedge D' & \quad\text{conjunction of definitions} \\
 & & & \\
J & ::= & & \text{join-patterns} \\
 & & x\langle\widetilde{y}\rangle & \quad\text{message pattern} \\
 & | & J \mid J' & \quad\text{join of patterns}
\end{array}
$$

Figure 1: Grammar of the join-calculus.

- $x\langle\widetilde{v}\rangle$ sends the tuple of values $\widetilde{v}$ on the channel named x. This message is asynchronous, in the sense that it does not require any form of handshake or acknowledgment.
- $\mathsf{def}\ D\ \mathsf{in}\ P$ is the process P in the scope of the local definitions given in D.
- $\mathsf{if}\ v = v'\ \mathsf{then}\ P\ \mathsf{else}\ P'$ tests whether $v = v'$, and then runs the process P or the process P' depending on the result of the test. (This construct, sometimes called matching, is an addition to the original syntax of the join-calculus.)
- $P \mid P'$ is the parallel composition of the processes P and P'.
- $\mathsf{0}$ is the null process, which does nothing.

A join-pattern is a non-empty list of message patterns, each of the form $x\langle y_1, \ldots, y_n\rangle$. The names $y_1, \ldots, y_n$ are bound, and should all be distinct. The name x is also bound; intuitively, it is the name of a channel being defined. Channels in the join-calculus are localized, in the sense that their names can be introduced only in join-patterns, like the name x. A join-pattern is much like a guard for a definition, in the sense that a definition $J \triangleright P$ says that the process P may run when there are messages that match the join-pattern J. (If there are messages that match the join-pattern J several times, then as many instances of P may run.) Next, we explain the notion of matching in a few special cases.

- Let us consider first the case where J is simply the join-pattern $x\langle y\rangle$. The join-pattern J is matched when a message v has been sent on x. When this happens, the message is consumed, and P is run with the actual argument v substituted for the formal argument y. (Thus, $x\langle y\rangle \triangleright P$ is analogous to the definition of a function with name x, formal argument y, and body P.)

- In the case where J is the join-pattern $x\langle y_1, \ldots, y_m\rangle$, we say that J is matched when a tuple $v_1, \ldots, v_m$ has been sent on x (with the same m). When this happens, the message is consumed, and P is run, with the actual arguments $v_1, \ldots, v_m$ substituted for the formal arguments $y_1, \ldots, y_m$.

- In the more general case where J is the join-pattern $x\langle y_1, \ldots, y_m\rangle \mid x'\langle y'_1, \ldots, y'_{m'}\rangle$, we say that J is matched when there are messages on both of the channels x and x', and these messages have m and m' components, respectively. When this happens, the messages are consumed, and P is run, with the actual arguments substituted for the corresponding formal arguments.

The reader may infer the full definition of matching from these special cases; for completeness, a full formal definition appears in Appendix A.

In addition to definitions of the form $J \triangleright P$, the grammar allows definitions of the form $D \wedge D'$. A definition $D \wedge D'$ is simply the conjunction of the definitions D and D'. A conjunction like $x\langle y\rangle \triangleright P \wedge x\langle z\rangle \triangleright Q$, where the same defined name x appears in two conjuncts, is legal; when there is a message $x\langle v\rangle$, either P or Q may run—the choice between them is non-deterministic.

Definitions obey lexical scoping rules. In particular, given a process $\mathsf{def}\ D\ \mathsf{in}\ P$, a channel name defined in D is recursively bound in the whole of $\mathsf{def}\ D\ \mathsf{in}\ P$, including D. Appendix A also contains a precise definition of the scoping rules.

Some variants of the pi-calculus have a primitive for replication; intuitively, the replication of a process P behaves like the parallel composition of infinitely many copies of P [37]. The join-calculus does not include such a primitive, because replication can be defined as an abbreviation:

$$\mathsf{repl}\ P \stackrel{\text{def}}{=} \mathsf{def}\ \kappa\langle\rangle \triangleright (P \mid \kappa\langle\rangle)\ \mathsf{in}\ \kappa\langle\rangle$$

where κ is a name that does not occur free in P.

Many additional control and data structures can be encoded in the join-calculus, but sometimes it is preferable to take those constructs as primitive.

In the main body of this paper, we give only the informal specifications of the constructs that we use; Appendix A contains precise specifications.

2.1.2 An example

As a first example, we consider the process:

$$P \stackrel{\text{def}}{=} \mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ x\langle u, v\rangle$$

Here, u and v are arbitrary values, z is the free name of a channel, x is the bound name of a channel, and y and y' are two other bound names that serve as formal parameters. The process P introduces the channel x, and attaches to it the definition $x\langle y, y'\rangle \triangleright z\langle y\rangle$. Thus, when a pair y, y' appears as a message on x, its first component is forwarded on z. The body of the process P is $x\langle u, v\rangle$, which sends the pair u, v on x.

Informally, we may think of x as a secure channel. Suppose that we put P in an arbitrary (and possibly hostile) context $C[\cdot]$. (As usual, a context is a process with a hole; if $C[\cdot]$ is a context, then $C[P]$ is the result of filling its hole with P.)

- In $C[P]$, the context $C[\cdot]$ cannot send on the channel x defined in P, so it cannot cause P to forward something other than u on z.
- In $C[P]$, the context $C[\cdot]$ cannot intercept the messages that travel on x in P, so in particular $C[\cdot]$ has no information at all about the value of v.

As in the pi-calculus and in the spi-calculus, such security properties are guaranteed by scoping, and they can be rephrased as equivalences between processes. Informally, we say that P_1 and P_2 are equivalent processes (and write $P_1 \approx P_2$) when no context can distinguish one from the other. For example, the equivalence:

$$\mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ x\langle u, v\rangle \ \approx\ z\langle u\rangle$$

equates P to a process that simply sends u on z, and it implies that the context of P cannot cause P to forward something other than u on z. Similarly, we may express the secrecy of v by the equivalence:

$$\mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ x\langle u, v\rangle \ \approx\ \mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ x\langle u, w\rangle$$

for all v and w. The two processes being equated are P and the result of replacing v with w in P. This equivalence means that the observable behavior of P does not depend on the value of v, even when P is put in an arbitrary—and possibly hostile—context. (Section 5 contains a precise definition of $\approx$, as observational equivalence.)

2.1.3 A larger example

We describe a larger example in some detail, both as explanation of the join-calculus and as additional motivation for our work. Our example concerns a contest. In its simplest variant, the contest relies on a server that creates a channel and publishes its name; the participant whose entry arrives first on this channel is the winner. At the level of abstraction of the join-calculus, it is relatively simple to understand the contest, to state its properties, and to consider variants, in great part because the join-calculus description of the contest hides challenging aspects of distributed communication (for example, cryptographic protection). We address some of those aspects in later sections.

For convenience in this example, we use strings as though they were primitive in the join-calculus, writing $+$ for string concatenation. Strings could easily be added to the join-calculus, but they can also be encoded through standard methods.

The following context essentially defines the contest:

$$\begin{array}{llllll} C[\,\cdot\,] & \stackrel{\text{def}}{=} & \mathsf{def} & \mathit{entry}\langle a, n\rangle \mid \mathit{open}\langle\rangle & \triangleright & a\langle n + \text{``}\mathit{won}\text{''}\rangle \mid \mathit{closed}\langle n\rangle \\ & & \wedge & \mathit{entry}\langle a', n'\rangle \mid \mathit{closed}\langle n\rangle & \triangleright & a'\langle n + \text{``}\mathit{won}\text{''}\rangle \mid \mathit{closed}\langle n\rangle \;\mathsf{in}\; [\,\cdot\,] \end{array}$$

The state of the contest is described by placing a process in $C[\,\cdot\,]$, for example as in:

$$C[\mathit{open}\langle\rangle \mid \mathit{plug}\langle \mathit{entry}\rangle]$$

and as in:

$$C[\mathit{closed}\langle \mathit{Alice}\rangle \mid \mathit{plug}\langle \mathit{entry}\rangle]$$

The channel *entry* is the one on which participants should send their entries. The channel *open* is local to the server; the presence of a message on this channel signifies that the contest is open. Similarly, the channel *closed* serves to represent that the contest is closed. (It is common to represent local state in this manner in process calculi.) The channel *plug* is free, and its only purpose is the publication of *entry*. The process $\mathit{open}\langle\rangle \mid \mathit{plug}\langle \mathit{entry}\rangle$ emits an empty message on the channel *open* and emits the name *entry* on the channel *plug*. When the server receives a first pair a, n on *entry*, it replies on a, with the string $n + \text{``}\mathit{won}\text{''}$. Thus, a is a continuation channel that represents a return address; n is arbitrary and represents the identity of the participant that sent the message. The server records n locally by producing the message $\mathit{closed}\langle n\rangle$ while consuming the message $\mathit{open}\langle\rangle$. Later, whenever the server receives another pair a', n' on *entry*, it replies on a' with the same string $n + \text{``}\mathit{won}\text{''}$, thus telling the new participant that an entry with the name n had priority. Thus, $C[\mathit{open}\langle\rangle \mid \mathit{plug}\langle \mathit{entry}\rangle]$ represents the starting state of the contest while $C[\mathit{closed}\langle \mathit{Alice}\rangle \mid \mathit{plug}\langle \mathit{entry}\rangle]$ represents a state where *Alice* has won.

In this example, an attacker cannot intercept or modify the entries in the contest or the server's replies. For instance, an attacker cannot modify the name n of the winner and cannot learn the name n' of a loser. We present some of these security properties, expressing them as equivalences; they follow from the scoping rules of the join-calculus and can be proved through standard bisimulation techniques.

- Only a participant with access to *entry* may win the contest. The following equivalence concerns the extreme case where in fact only one participant has access to *entry*:

$$C[\mathit{open}\langle\rangle \mid \mathit{entry}\langle a, \mathit{Alice}\rangle] \quad \approx \quad C[\mathit{closed}\langle \mathit{Alice}\rangle \mid a\langle \mathit{Alice}+\text{``}\mathit{won}\text{''}\rangle]$$

- The outcome of the contest cannot be affected once the contest is closed:

$$\begin{aligned} & C[\mathit{closed}\langle \mathit{Alice}\rangle \mid \mathit{plug}\langle \mathit{entry}\rangle] \\ \approx \quad & \mathsf{def}\ \mathit{entry}\langle a, n\rangle \triangleright a\langle \mathit{Alice}+\text{``}\mathit{won}\text{''}\rangle\ \mathsf{in}\ \mathit{plug}\langle \mathit{entry}\rangle \end{aligned}$$

  The latter process behaves monotonously, just like the contest after closing.

- A loser remains anonymous with respect to other participants:

$$\begin{aligned} & C[\mathit{closed}\langle \mathit{Alice}\rangle \mid \mathit{plug}\langle \mathit{entry}\rangle \mid \mathit{entry}\langle a', \mathit{Bob}\rangle] \\ \approx \quad & C[\mathit{closed}\langle \mathit{Alice}\rangle \mid \mathit{plug}\langle \mathit{entry}\rangle \mid \mathit{entry}\langle a', \mathit{Patrick}\rangle] \end{aligned}$$

  This equivalence means that the outer context cannot distinguish an entry that contains *Bob* and an entry that contains *Patrick*. The two processes being compared are in fact equivalent to:

$$C[\mathit{closed}\langle \mathit{Alice}\rangle \mid \mathit{plug}\langle \mathit{entry}\rangle \mid a'\langle \mathit{Alice}+\text{``}\mathit{won}\text{''}\rangle]$$

  which is independent of *Bob* and *Patrick*.

The contest server may spread its work over auxiliary components (proxies), as follows. Each of the proxies makes available a distinct channel entry_i for entries, forwards the first entry that it receives to the server, waits for an answer, and remembers this answer so that it can immediately return it in response to later entries. The following context describes proxy i:

$$\begin{array}{llll} C_i[\,\cdot\,] \ \stackrel{\text{def}}{=} & \mathsf{def} & \mathit{entry}_i\langle a, n\rangle \mid \mathit{open}_i\langle\rangle & \triangleright\ \mathit{entry}\langle a, n\rangle \mid \mathit{entry}\langle \mathit{closed}_i, n\rangle \\ & \wedge & \mathit{entry}_i\langle a', n'\rangle \mid \mathit{closed}_i\langle s\rangle & \triangleright\ a'\langle s\rangle \mid \mathit{closed}_i\langle s\rangle\ \mathsf{in}\ [\,\cdot\,] \end{array}$$

The main difference between a proxy and the server is that, when a proxy receives its first entry $\mathit{entry}_i\langle a, n\rangle$, it cannot decide by itself whether this entry is the winning one. The proxy leaves this decision to the server by submitting two entries with the same name n but with different return addresses. The message $entry\langle a, n\rangle$ will lead to a direct response on a to participant n; in effect, this message extends the capability to send messages on a to the server. On the other hand, the message $entry\langle closed_i, n\rangle$ will result in an identical response on $closed_i$ to the proxy. The proxy remembers this response, so it can handle subsequent entries without consulting the server.

Combining the contexts $C[\,\cdot\,]$ and $C_i[\,\cdot\,]$, we can assemble a system with a main server and two proxies:

$$C\Big[open\langle\rangle \mid C_1[open_1\langle\rangle \mid plug_1\langle entry_1\rangle] \mid C_2[open_2\langle\rangle \mid plug_2\langle entry_2\rangle]\Big]$$

This system shares the essential security properties of the system without proxies (for instance, the anonymity of losers). The question of whether the proxies may misbehave does not arise: since we have their code, we can verify them.

Intuitively, the main server, the proxies, and the participants may be located at different sites. A distributed implementation of the contest may therefore rely on some amount of cryptography. Obviously, the implementation could protect the channels between the sites with cryptographic protocols. In addition, the implementation could represent channel names as unguessable capabilities, so only the processes that obtain a return address as the result of legitimate communication can send messages to the address. Like other high-level programming languages, the join-calculus abstracts away these difficult details of distributed communication. We reconsider these details in section 6.2.

2.2 The sjoin-calculus

Our low-level language is an extension of the join-calculus. This low-level language includes operations that model encryption and decryption; in addition, it includes names that can be used as keys, nonces, or other tags. We call this low-level language the sjoin-calculus (by analogy with the spi-calculus).

It is convenient that the sjoin-calculus be an extension of the join-calculus, but it is not actually necessary. We do not rely on the trivial translation from the join-calculus to the sjoin-calculus, which maps every process to itself. The sjoin-calculus processes that our translation produces rely on cryptographic protection for communication across machines, rather than on the abstract secure channels provided by the join-calculus.

2.2.1 Syntax and informal semantics

The grammar of the sjoin-calculus extends that of the join-calculus; it is given in Figure 2. The set of values includes not only names but also ciphertexts of the form $\{\widetilde{v}\}_v$. In the ciphertext $\{\widetilde{v}\}_v$, the subscript v is the encryption key, and $\widetilde{v}$ is

v	::=		values
		x	name
	$\vert$	$\{\widetilde{v}\}_v$	encryption
P	::=		processes
		$x\langle\widetilde{v}\rangle$	message
	$\vert$	$\mathsf{def}\ D\ \mathsf{in}\ P$	local definition
	$\vert$	$\mathsf{if}\ v = v'\ \mathsf{then}\ P\ \mathsf{else}\ P'$	comparison
	$\vert$	$\mathsf{decrypt}\ v\ \mathsf{using}\ v'\ \mathsf{to}\ \widetilde{x}\ \mathsf{in}\ P\ \mathsf{else}\ P'$	decryption
	$\vert$	$P \mid P'$	parallel composition
	$\vert$	$\mathsf{0}$	null process
D	::=		definitions
		$J \triangleright P$	reaction rule
	$\vert$	$\mathsf{fresh}\ x$	fresh name
	$\vert$	$\mathsf{keys}\ x^+, x^-$	fresh pair of keys
	$\vert$	$D \wedge D'$	conjunction of definitions
J	::=		join-patterns
		$x\langle\widetilde{y}\rangle$	message pattern
	$\vert$	$J \mid J'$	join of patterns

Figure 2: Grammar of the sjoin-calculus.

the cleartext. We adopt a simple, black-box model of cryptographic operations (cf. [27]). Informally, we assume that the cleartext or the encryption key cannot be extracted from the ciphertext without prior knowledge of a corresponding decryption key, that is, of the inverse of the encryption key. We also assume that prior knowledge of a corresponding decryption key reveals the cleartext, and that the cleartext contains sufficient redundancy so that successful decryption is evident. Ciphertexts can be compared, as for example in the process $\mathsf{if}\ \{v\}_x = \{v'\}_x\ \mathsf{then}\ P\ \mathsf{else}\ P'$; this comparison would enable a recipient of $\{v\}_x$ with knowledge of x and v' to deduce whether v equals v'.

The syntax of processes includes a new decryption form. The process $\mathsf{decrypt}\ v\ \mathsf{using}\ v'\ \mathsf{to}\ \widetilde{x}\ \mathsf{in}\ P\ \mathsf{else}\ P'$ attempts to decrypt v using v' as decryption key. If this decryption succeeds, then P runs, with the results of the decryption substituted for $\widetilde{x}$; if the decryption fails, then P' runs. For example, if y^+ is the name of an encryption key and y^- is the name of the inverse decryption key, then the process

$$\mathsf{decrypt}\ \{z\}_{y^+}\ \mathsf{using}\ y^-\ \mathsf{to}\ x\ \mathsf{in}\ P\ \mathsf{else}\ P'$$

will decrypt $\{z\}_{y^+}$ using y^-, and will run P with z substituted for x.

The syntax of definitions includes constructs for introducing names and pairs of keys. The construct $\mathsf{fresh}\ x$ introduces the fresh name x. The construct $\mathsf{keys}\ x^+, x^-$ introduces the names of keys x^+ and x^-; the key x^+ is an encryption key, and x^- is the inverse decryption key. Note that x^+ and x^- range over ordinary names: the symbols $+$ and $-$ are used only conventionally, and have no formal meaning. In general, the names x^+ and x^- will be different; this distinction models the separation of encryption key and decryption key in public-key cryptosystems (e.g., [33]). However, we do not require that x^+ and x^- be distinct; we use $\mathsf{keys}\ x, x$ when x is a shared key (so x is its own inverse), and abbreviate $\mathsf{keys}\ x, x$ to $\mathsf{key}\ x$. This notation yields a concise model of shared-key cryptography as a special case of public-key cryptography.

An arbitrary value v may be used as an encryption key, in the form $\{\widetilde{v}\}_v$, but an inverse for v would exist only if v is a name introduced with the construct keys. Thus, we do not consider methods for generating a key pair from a ciphertext, from a channel name, or from an arbitrary fresh name. Such methods arise in practical protocols, but we do not need them in our translation of the join-calculus.

The syntax of join-patterns is exactly as in the join-calculus.

Given this context-free grammar, one can write silly expressions, for example $\mathsf{def}\ \mathsf{keys}\ x, x\ \mathsf{in}\ x\langle y\rangle$ where a shared key x is used as a channel. A straightforward extension of the type system of the join-calculus excludes such expressions. This extension consists in adding a basic type $\mathsf{BitString}$, with the rules that the names introduced by the constructs $\mathsf{fresh}\ x$ and $\mathsf{keys}\ x^+, x^-$ are of type $\mathsf{BitString}$, and that encryption and decryption operations apply only to arguments of type $\mathsf{BitString}$ and yield results of type $\mathsf{BitString}$.

2.2.2 An example

Our first example for the sjoin-calculus continues our first example for the join-calculus, which concerns the process:

$$P \stackrel{\text{def}}{=} \mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ x\langle u, v\rangle$$

When this process is implemented on a single machine, the local operating system can guarantee the desired security properties for the channel x. On the other hand, suppose that the definition $x\langle y, y'\rangle \triangleright z\langle y\rangle$ and the body $x\langle u, v\rangle$ are located in different machines. In that case, the channel x might be implemented by a TCP connection between the machines. However, this implementation would be subject to a variety of attacks.

Encryption can be used in order to thwart those attacks. In particular, the implementation of the channel x may associate a key with x, and encrypt messages on x using this key. The resulting implementation of P might be described in the sjoin-calculus as follows:

$$P' \stackrel{\text{def}}{=} \mathsf{def\ keys}\ x^+, x^- \wedge a\langle c\rangle \triangleright (\mathsf{decrypt}\ c\ \mathsf{using}\ x^-\ \mathsf{to}\ y, y'\ \mathsf{in}\ z\langle y\rangle\ \mathsf{else}\ 0)\ \mathsf{in}\ e\langle a, \{u, v\}_{x^+}\rangle$$

Our intent is that the expressions $a\langle c\rangle \triangleright (\mathsf{decrypt}\ c\ \mathsf{using}\ x^-\ \mathsf{to}\ y, y'\ \mathsf{in}\ z\langle y\rangle\ \mathsf{else}\ 0)$ and $e\langle a, \{u, v\}_{x^+}\rangle$ may be located in different machines. The name e represents a public, unprotected channel that the process P' uses (like an Ethernet). Instead of transmitting u and v on a private channel x, the process P' transmits them on the public channel e, but after encryption under the key x^+ and tagging with the name a. It is the responsibility of the environment to relay $\{u, v\}_{x^+}$ on a; the environment uses a as an address, for routing $\{u, v\}_{x^+}$. On receipt of a message c on a, the expression

$$a\langle c\rangle \triangleright (\mathsf{decrypt}\ c\ \mathsf{using}\ x^-\ \mathsf{to}\ y, y'\ \mathsf{in}\ z\langle y\rangle\ \mathsf{else}\ 0)$$

attempts to decrypt a pair y, y' using x^- as decryption key and, in case of success, forwards the first component of the pair on z.

While this process P' may resemble a secure distributed implementation of P, because it uses encryption, it still has several gaps. For example, nothing prevents an attacker from replaying the message $\{u, v\}_{x^+}$ on a. Whereas P' would then output u on z twice, P outputs u on z only once. Similarly, nothing prevents an attacker from intercepting all messages on e, thus causing the absence of any output on z. A more subtle problem is that the process P' permits traffic analysis while P does not, in the sense that an attacker may notice that some communication takes place on e in P', while P is perfectly silent to the outside until the output on z.

The techniques that we describe in this paper address these difficulties. We show a systematic, mechanical way of implementing a process such as P by translating it to a lower-level process similar to P'. Like P', the lower-level processes that we construct rely on cryptographic protection for communication across machines rather than on abstract secure channels. On the other hand, our translation is more sophisticated than P' might suggest. It protects against various attacks and—at least within our formal framework—we can prove that it is correct.

In particular, we can prove that if P_1 and P_2 are two equivalent processes of the join-calculus, then their implementations are equivalent processes of the sjoin-calculus, and vice versa. As an example, consider the join-calculus equivalence:

$$\mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ x\langle u, v\rangle \;\approx\; \mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ x\langle u, w\rangle$$

which expresses that P does not reveal v. Our results imply that the implementation of P does not reveal v either.

In the simplest case, our translation assumes that every message may cross machine boundaries, and therefore that every message requires cryptographic protection. Our translation technique is rather flexible, so it enables us to treat more sophisticated cases, where some communication is local to a machine or takes place on a trusted network. In those cases, only messages that cross trust boundaries are encrypted; the basic component of the translation is a process that operates much as a firewall with an encrypting tunnel (e.g., [12, 7, 2]), relaying messages from a trusted machine or a trusted network to a public network and conversely, applying encryption and converting message formats through appropriate marshaling and unmarshaling procedures.

2.2.3 On the choice of the join-calculus and the sjoin-calculus

The problem of implementing communication constructs securely is not unique to the join-calculus. This problem has arisen naturally in other settings, for example in the context of network-object systems [43] (see also [44, 42]) and in the context of the pi-calculus and the spi-calculus [6, section 8].

While it may be possible to address the problem in the context of network-object systems, doing this precisely seems hard because of the complexity of those systems and the lack of a formal semantics for them. In contrast, the join-calculus and the sjoin-calculus are simple enough that we can develop a satisfactory theory for them, yet rich enough to present interesting security issues.

Another advantage of the join-calculus as the starting point for our work is that it relies on asymmetric (one-directional) channels, unlike the standard pi-calculus which relies on symmetric (two-directional) channels. We did in fact attempt to use the pi-calculus, but we encountered a subtle difficulty [1].

- Consider a pi-calculus process R that creates a channel c, uses it for sending and receiving an internal message v, and finally sends the name c on a public channel. In the pi-calculus, a process that receives the name c is not able to work back and read the earlier traffic on c. This property is a form of forward secrecy [16].

- On the other hand, a natural implementation of R would map the channel c to a shared key k. An attacker may log messages that are transmitted on the public network and, if it ever receives k, the attacker may decrypt earlier messages under k. Thus, the natural implementation of R does not offer forward secrecy, and may leak v.

This problem does not arise in the join-calculus because, in the join-calculus, the capability of receiving on a channel can never be transferred. Abstractly, the use of the join-calculus instead of the pi-calculus does not entail a loss of expressiveness, since there is an encoding of the pi-calculus in the join-calculus [18]. We can compose that encoding with our translation from the join-calculus to the sjoin-calculus, obtaining a cryptographic implementation of the pi-calculus. However, the resulting implementation is significantly more complex and inefficient than for the join-calculus.

3 Low-level protocols for communication

In this section, we discuss our model of low-level communication, and produce two secure protocols that transmit a single message. A third protocol appears in a conference paper [5]. Any of these protocols, or any other protocol with the same properties, can be used in the translation of section 4.

3.1 The low-level network model

Our informal assumptions about the network used by sjoin-calculus processes are similar to those of Needham and Schroeder [40]:

> We assume that an intruder can interpose a computer in all communication paths, and thus can alter or copy parts of messages, replay messages, or emit false material.
>
> We also assume that each principal has a secure environment in which to compute, such as is provided by a personal computer or would be by a secure shared operating system.

Like Needham and Schroeder, we are not concerned about timing-analysis attacks. Timing-analysis attacks are outside the scope of our calculi (which are asynchronous, and do not include any notion of global time).

In our model, anyone can send and receive messages of type $\mathsf{BitString}$ through the network interface consisting of the public channels *emit* and *recv*. For output, a process P sends its message on *emit*. For input, P sends a continuation channel κ of type $\langle\mathsf{BitString}\rangle$ on *recv*; the network will then return a single message on κ. This behavior of the network is described by the expression:

$$\mathsf{def}\ recv\langle\kappa\rangle \mid emit\langle m\rangle \triangleright \kappa\langle m\rangle\ \mathsf{in}\ P$$

Abstracting from this expression, we obtain the context $\mathcal{E}[\,\cdot\,]$:

$$\mathcal{E}[\,\cdot\,] \quad \stackrel{\text{def}}{=} \quad \mathsf{def}\ recv\langle\kappa\rangle \mid emit\langle m\rangle \triangleright \kappa\langle m\rangle\ \mathsf{in}\ [\cdot]$$

Since calls to $recv\langle\kappa\rangle$ retrieve messages from the network non-deterministically, processes may have to filter for messages that are destined for them. We rely on keys as the basis for this filtering, with the following definition:

$$\begin{array}{lcll} \mathsf{let}\ \widetilde{y} = \mathsf{filter}\ k\ \mathsf{in}\ P & \stackrel{\text{def}}{=} & \mathsf{def}\ \kappa\langle m\rangle \triangleright & \mathsf{decrypt}\ m\ \mathsf{using}\ k\ \mathsf{to}\ \widetilde{y}\ \mathsf{in}\ P \\ & & & \mathsf{else}\ emit\langle m\rangle \mid recv\langle\kappa\rangle\ \mathsf{in} \\ & & recv\langle\kappa\rangle & \end{array}$$

This process picks a message m from the network and attempts to decrypt it using a key k. If the decryption succeeds, then the process executes P with m's contents substituted for $\widetilde{y}$; otherwise, the process returns m to the network and retries. This method of addressing is fairly rudimentary and unrealistic on an Ethernet and even more on a wide-area network: every message must be broadcast to all potential recipients, and receipt of a broadcast implies a non-trivial amount of computation for decryption. Unfortunately, using cleartext addresses would expose us to a subtle traffic-analysis attack.

Because of the simplicity of our network interface, injecting noise into the network in order to prevent traffic analysis is straightforward. The process W creates the desired noise:

$$W \stackrel{\text{def}}{=} \mathsf{repl}\ (\mathsf{def}\ \mathsf{fresh}\ m\ \mathsf{in}\ \mathsf{repl}\ emit\langle m\rangle)$$

The process W endlessly generates endlessly repeated fresh messages.

Whenever we study the implementations of join-calculus processes, we do it in the context $\mathcal{E}[\,\cdot\,]$ and in the presence of W, and in addition in the presence of the process $plug\langle emit, recv\rangle$. This process transmits the network-interface channels $emit$ and $recv$ on a public channel $plug$, so that any outside process has access to them. Combining these expressions, we obtain the following context:

$$\mathcal{E}nv\,[\,\cdot\,] \stackrel{\text{def}}{=} \mathcal{E}\,[\,plug\langle emit, recv\rangle \mid W \mid \cdot\,]$$

Note that, while an intruder may intercept any message, we do not guarantee that it actually will intercept every message. (In this we seem to differ from Needham and Schroeder.) The intruder has access to the channels $emit$ and $recv$, but cannot control their behavior. This assumption is fairly realistic, and simplifies our task. For example, the translations of the processes $x\langle\rangle$ and $\mathsf{def}\ y\langle\rangle \triangleright x\langle\rangle\ \mathsf{in}\ y\langle\rangle$ should be indistinguishable, since these processes are indistinguishable according to the semantics of the join-calculus. On the other hand, an intruder that was in charge of transporting every message could observe a difference between $x\langle\rangle$ and $\mathsf{def}\ y\langle\rangle \triangleright x\langle\rangle\ \mathsf{in}\ y\langle\rangle$ by interrupting all messages on y, then observing whether a message on x appears anyway. There would however be no harm in guaranteeing that an intruder can obtain a copy of every message. Formally, we could provide this guarantee by changing the definition of $\mathcal{E}[\,\cdot\,]$, as follows:

$$\mathcal{E}\,[\,\cdot\,] \stackrel{\text{def}}{=} \mathsf{def} \begin{array}{ll} & emit\langle m\rangle \triangleright data\langle m\rangle \mid leak\langle m\rangle \\ \wedge & recv\langle\kappa\rangle \mid data\langle m\rangle \triangleright \kappa\langle m\rangle \end{array} \mathsf{in}\ [\cdot]$$

Our results would still hold with this definition, with only minor changes.

3.2 Translation framework

To implement the join-calculus in terms of the sjoin-calculus, we map each channel x to a key pair x^+, x^-. However, we cannot simply map each message $\widetilde{v}$ on x to a message $\{\widetilde{v}\}_{x^+}$ encrypted with x^+ on *emit*, to be received on *recv* and decrypted with x^-. The resulting sjoin-calculus process would be subject to attacks that are not possible on the original join-calculus process. For example, an eavesdropper might recognize a message $\{\widetilde{v}\}_{x^+}$ if it already knows x^+ and $\widetilde{v}$, encodes $\widetilde{v}$ under x^+, and compares the resulting ciphertext against the message. Adding a fresh component to the message (a confounder) would prevent this small leak. Also, an active intruder could replay $\{\widetilde{v}\}_{x^+}$ later; several common techniques can prevent this replay attack, for example the use of timestamps.

In sections 3.3 and 3.4 we present two alternative methods for implementing a message on the channel x. In both cases, we map a message on x to an execution of a protocol that uses the keys x^+, x^- and the channels *emit* and *recv*. Although the two protocols have significant differences, either is adequate for our purposes. Both of the protocols are based on two sjoin-calculus processes:

- $E_x[\widetilde{v}]$ sends a tuple $\widetilde{v}$ of values, using the encryption key x^+.
- R_x uses the key x^- to decrypt a message, and forwards its contents in clear on an internal channel x°.

Naively, one may try the following definitions of $E_x[\widetilde{v}]$ and R_x:

$$\begin{aligned} E_x[\widetilde{v}] &\stackrel{\text{def}}{=} \mathit{emit}\langle\{\widetilde{v}\}_{x^+}\rangle \\ R_x &\stackrel{\text{def}}{=} \mathsf{let}\ \widetilde{y} = \mathsf{filter}\ x^-\ \mathsf{in}\ x^\circ\langle\widetilde{y}\rangle \end{aligned}$$

where the length of the tuple $\widetilde{y}$ in the definition of R_x is deduced from the type of x. We refer to the protocol that uses these definitions as the naive protocol; as explained above, the naive protocol permits attacks. The protocols of sections 3.3 and 3.4 require more sophisticated definitions of $E_x[\widetilde{v}]$ and R_x.

Throughout, we associate three distinct names x^+, x^-, and x° with each channel name x; both x^+ and x^- are of type $\mathsf{BitString}$ while x° is of type $\langle\mathsf{BitString}, \ldots, \mathsf{BitString}\rangle$ with the same arity as x. For every name $y \neq x$, we assume that x^+, x^-, and x° are distinct from y^+, y^-, and y°, and do not appear in the processes $E_y[\,\cdot\,]$ and R_y.

3.3 Basic protocol

The basic protocol is an enhancement of the naive protocol in several respects:

- The emitter of a message repeats the message indefinitely, in case the message is intercepted.

- The emitter also adds a fresh component c to each message. This component serves both as a confounder (preventing the generation and detection of two identical messages) and as a unique identifier (for thwarting replay attacks).

- The recipient of a message extracts and records its unique identifier, and drops any message whose unique identifier has been recorded previously.

The last point implies that the recipient must maintain a data structure with unique identifiers. We omit the details of how to program such a data structure in the sjoin-calculus. We simply write the declaration $\mathsf{uids}\ t$ to denote the definition of an initially empty set of unique identifiers t; and write $\mathsf{if\ not\ tset}\ t(c)\ \mathsf{then}\ P$ for a process that atomically tests whether the unique identifier c is in t, and if not adds c to t and then triggers the execution of P. (The "test-and-set" must be atomic, but the execution of P need not be.)

We arrive at the following definitions.

$$
\begin{aligned}
E_x[\widetilde{v}] &\stackrel{\text{def}}{=} \mathsf{def\ fresh}\ c\ \mathsf{in\ repl}\ emit\langle\{c,\widetilde{v}\}_{x^+}\rangle \\
F_x &\stackrel{\text{def}}{=} \mathsf{let}\ c,\widetilde{y} = \mathsf{filter}\ x^-\ \mathsf{in\ if\ not\ tset}\ t_x(c)\ \mathsf{then}\ x^\circ\langle\widetilde{y}\rangle \\
R_x &\stackrel{\text{def}}{=} \mathsf{def\ uids}\ t_x\ \mathsf{in\ repl}\ F_x
\end{aligned}
$$

where the length of the tuple $\widetilde{y}$ in the definition of F_x is deduced from the type of x. The process F_x is a component of the recipient that attempts to receive a single message; the recipient simply replicates such attempts and defines a shared set of unique identifiers.

The basic protocol has a major drawback: the recipients of messages must remember the unique identifiers of those messages for ever, and in principle need unbounded memory. The session-based protocol, presented next, avoids this drawback by relying on a challenge-response dialogue.

3.4 Session-based protocol

In the session-based protocol, each join-calculus message on a channel x is mapped to a three-message dialogue that relies on the keys x^+ and x^-. In order to prevent attacks, all three messages are encrypted.

- The emitter first sends a request for a challenge to the recipient. The request is a new shared key k encrypted with x^+. To guard against message loss, the emitter sends this request repeatedly, and selects a single challenge to proceed with the protocol.

- When the recipient receives such a request, it returns a challenge and waits for a single response. The challenge is a new shared key k' encrypted with k.

- The actual payload data is transmitted in the final message. It is encrypted with k'.

The resulting definitions are as follows.

$$
\begin{aligned}
E_x[\widetilde{v}] &\stackrel{\text{def}}{=} \textsf{def key } k \textsf{ in} \\
&\quad (\textsf{repl } emit\langle\{k\}_{x^+}\rangle) \mid \textsf{let } k' = \textsf{filter } k \textsf{ in } (\textsf{repl } emit\langle\{\widetilde{v}\}_{k'}\rangle) \\
F_x &\stackrel{\text{def}}{=} \textsf{let } k = \textsf{filter } x^- \textsf{ in} \\
&\quad \textsf{def key } k' \textsf{ in } ((\textsf{repl } emit\langle\{k'\}_k\rangle) \mid \textsf{let } \widetilde{y} = \textsf{filter } k' \textsf{ in } x^\circ\langle\widetilde{y}\rangle) \\
R_x &\stackrel{\text{def}}{=} \textsf{repl } F_x
\end{aligned}
$$

The key k' can be regarded as a session key; the steps where the emitter sends k under x^+ and the recipient replies with k' under k constitute a key-establishment protocol for a session. Here the session conveys a single useful payload, $\widetilde{v}$. However, it would be straightforward to extend the protocol to longer dialogues. One may also use this protocol to establish site-to-site sessions, and then multiplex channels on these sessions; this technique would save on public-key operations, but would leak some information about the distribution of channels across sites.

Like the basic protocol, the session-based protocol has some unrealistic elements. In particular, the session-based protocol does not include any time-outs; after producing a challenge, the challenger waits for a response indefinitely. Moreover, the protocol produces a steady stream of encrypted traffic, even after the payload has reached the recipient. These aspects of the protocol simplify our study, for example saving us from coding and reasoning about acknowledgments. It would be a "simple matter of programming" to obtain a more realistic protocol; however, it may be quite hard to reason about the resulting design.

4 Translations

Section 3 describes sjoin-calculus protocols that can be used as implementations for single join-calculus messages. Since message-passing is the fundamental computation step in the join-calculus, these protocols are the basis for complete implementations of the join-calculus in the sjoin-calculus. In this section we define two techniques for doing these implementations.

- The first technique is a compositional translation from the join-calculus to the sjoin-calculus. This translation replaces every join-calculus communication step with an execution of one of our protocols. It has a direct inductive definition, where for example the parallel composition of two processes is mapped to the parallel composition of their respective translations.

- The second technique places a join-calculus process in a context that filters every communication with the environment. This context is analogous to a firewall with an encrypting tunnel.

In section 5, we relate these two techniques, showing that they are in a sense equivalent (Theorem 3). Both of the techniques work equally well when based on the basic protocol of section 3.3 or the session-based protocol of section 3.4.

4.1 A translation where every message is encrypted

Our first technique does not make any assumptions about the distribution of the process being translated. In principle, every message could cross machine boundaries or Intranet boundaries, and could travel on a physically insecure network. Therefore, our technique protects every message of the source process by implementing it with a cryptographic protocol, whether the message is internal to the process or is exchanged between the process and its environment. The security properties of the translation do not depend on any notion of security perimeter (of "inside" and "outside").

4.1.1 Definition

If P is a join-calculus process, then we write $[\![P]\!]$ for its translation to the sjoin-calculus; we write $[\![\cdot]\!]$ for the translation function, and define it inductively as follows. On processes, we define $[\![\cdot]\!]$ by:

$$\begin{array}{rcl}
[\![x\langle\widetilde{v}\rangle]\!] & \stackrel{\text{def}}{=} & E_x[\widetilde{v}^+] \\
[\![\mathsf{def}\ D\ \mathsf{in}\ P]\!] & \stackrel{\text{def}}{=} & \mathsf{def}\ \bigwedge_{x\in\mathsf{dv}[D]}\mathsf{keys}\ x^+, x^-\ \mathsf{in} \\
 & & \left(\mathsf{def}\ [\![D]\!]\ \mathsf{in}\ \prod_{x\in\mathsf{dv}[D]} R_x\right) \mid [\![P]\!] \\
[\![\mathsf{if}\ v = v'\ \mathsf{then}\ P\ \mathsf{else}\ P']\!] & \stackrel{\text{def}}{=} & \mathsf{if}\ v^+ = v'^+\ \mathsf{then}\ [\![P]\!]\ \mathsf{else}\ [\![P']\!] \\
[\![P \mid P']\!] & \stackrel{\text{def}}{=} & [\![P]\!] \mid [\![P']\!] \\
[\![\mathsf{0}]\!] & \stackrel{\text{def}}{=} & \mathsf{0}
\end{array}$$

where $\widetilde{v}^+$ is $v_1^+, \ldots, v_n^+$ if $\widetilde{v}$ is $v_1, \ldots, v_n$, and where $\mathsf{dv}[D]$ is the set of names defined in D and $\prod_{x\in\mathsf{dv}[D]} R_x$ is the parallel composition of all processes R_x for $x \in \mathsf{dv}[D]$.

On join-patterns and definitions, we define $[\![\cdot]\!]$ by:

$$\begin{array}{rclcrcl}
[\![x\langle\widetilde{y}\rangle]\!] & \stackrel{\text{def}}{=} & x^\circ\langle\widetilde{y}^+\rangle & \qquad & [\![J \triangleright P]\!] & \stackrel{\text{def}}{=} & [\![J]\!] \triangleright [\![P]\!] \\
[\![J \mid J']\!] & \stackrel{\text{def}}{=} & [\![J]\!] \mid [\![J']\!] & \qquad & [\![D \wedge D']\!] & \stackrel{\text{def}}{=} & [\![D]\!] \wedge [\![D']\!]
\end{array}$$

By construction, this translation has the property that $[\![C[P]]\!] = [\![C]\!][[\![P]\!]]$ for every context $C[\cdot]$ and process P. This property is a formal reflection of the absence of any assumptions about distribution. The translation yields processes that can be executed in a distributed manner without any "built-in" security properties of channels. In $[\![P \mid P']\!]$, the components $[\![P]\!]$ and $[\![P']\!]$ may be located in different machines. Similarly, in $[\![\mathsf{def}\ D\ \mathsf{in}\ P]\!]$, the components $[\![P]\!]$ and $\mathsf{def}\ [\![D]\!]\ \mathsf{in}\ \prod_{x \in \mathsf{dv}[D]} R_x$ may be located in different machines. Except for a few continuations for internal use, channels convey only encrypted messages and encryption keys.

When x is a free name in the source process, the encryption key x^+ is free in the target process. Such free encryption keys and the network interface $emit, recv$ are the only free names in the processes that the translation produces (in addition to the name $plug$, which occurs in $\mathcal{E}nv\,[\,\cdot\,]$). The initial scope of the names x° and x^- contains only the protocol R_x, and these names are never transmitted. Thus, the scope of these names is purely static. In contrast, the name x^+ may be included in messages, and sent outside the static scope of the definition $\mathsf{keys}\ x^+, x^-$ where x^+ is introduced.

4.1.2 An example

As an example, we translate the simple process considered in section 2.1.2. According to the definitions, $[\![\mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ x\langle u, v\rangle]\!]$ is:

$$\mathsf{def}\ \mathsf{keys}\ x^+, x^-\ \mathsf{in}\ ((\mathsf{def}\ x^\circ\langle y^+, y'^+\rangle \triangleright E_z[y^+]\ \mathsf{in}\ R_x) \mid E_x[u^+, v^+])$$

The process obtained through this translation relies on the keys x^+ and x^- and on the channel x°, which correspond to the channel x of the original process. The subprocess $E_x[u^+, v^+]$ sends the keys u^+, v^+, encrypted with x^+, on $emit$. The subprocess R_x may pick up this message through $recv$, decrypt it using x^-, and forward its contents on x°. The definition $x^\circ\langle y^+, y'^+\rangle \triangleright E_z[y^+]$ implies that, if u^+, v^+ are received on x°, then u^+ is resent on $emit$, now encrypted with z^+.

As stated in section 2.1.2, the process $\mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ x\langle u, v\rangle$ is equivalent to the process $z\langle u\rangle$. The translation of $z\langle u\rangle$ is simply $E_z[u^+]$, which sends u^+ on $emit$ encrypted with z^+.

In light of this example, one may expect that the translation preserves equivalence, provided the channels $emit$ and $recv$ function as expected. More precisely, one may expect that if $P \approx Q$ then $\mathcal{E}nv\,[\,[\![P]\!]\,] \approx \mathcal{E}nv\,[\,[\![Q]\!]\,]$. This implication does indeed hold; it is proved in section 5.

4.2 A filtering translation

An implementation of the join-calculus that does not assume anything about distribution may be elegant and simple, but it may also be rather wasteful: it requires encryption even for messages internal to a protected site (a machine or a network). Next, we describe an implementation that avoids encryption in those cases.

Unlike that of section 4.1, the translation that we describe here is not defined inductively. Instead, the translation of a process P is simply P put in a suitable context. This context serves as a wrapper, or filter. It does not disturb messages internal to P. It does however catch messages that cross the boundary of P, adding or removing encryption, and translating contents (marshaling and unmarshaling). For example, a message that would leave P on a channel x gets turned into a $\mathsf{BitString}$ message encrypted with the key x^+; if the message contains the channel name v, then v gets replaced with the key v^+. Conversely, when a message arrives at the filter encrypted with a key y^+ and the filter has the key y^-, the contents of the message are translated and relayed on a channel y.

This filter resembles a firewall with an encrypting tunnel, so we sometimes refer to it as a firewall (a little abusively, perhaps). From the programmer's point of view, this filter serves as a run-time system for distributed communication that takes care of encryption and decryption and of marshaling and unmarshaling (e.g., [43]).

4.2.1 Data structures

Our firewall keeps track of the correspondence between encryption keys on the outside and channel names on the inside of a process. This correspondence is recorded in association tables. These association tables are auxiliary data structures that can be encoded in the join-calculus and a fortiori in the sjoin-calculus. We omit their encoding, and describe only their interface.

- The definition $\mathsf{assoc}\ S,\ t(x) = T,\ t'(x^+) = T'$ introduces an association table with content S. This definition binds two lookup functions t and t', and attaches the processes T and T' to them.

- The content S is a finite set of pairs (x, v) where x is a channel name and v is a value of type $\mathsf{BitString}$.

- In a definition $\mathsf{assoc}\ S,\ t(x) = T,\ t'(x^+) = T'$ of an association table, the processes T and T' are both of the form $\mathsf{def}\ D\ \mathsf{in}\ Q \mid \mathsf{enter}\ x, x^+$. The process $\mathsf{enter}\ x, x^+$, which appears once in T and once in T', has the role of entering the association between x and x^+ in the table, adding the pair (x, x^+) to S. In T, x^+ is defined in D while x is free. In T', conversely, x is defined in D while x^+ is free.

- In the scope of a definition $\mathsf{assoc}\ S,\ t(x) = T,\ t'(x^+) = T'$, the process $\mathsf{let}\ x^+ = t(v)\ \mathsf{in}\ P$ looks for a key associated with v in S. If one is found, then P is executed with the key substituted for x^+. Otherwise, the process T attached to t is executed; it creates a key and enters the association between v and this key. In parallel, P is executed with the key substituted for x^+.

- Similarly, the process $\mathsf{let}\ x = t'(v')\ \mathsf{in}\ P'$ looks for a channel name associated with v' in S and, if one is not found, creates a channel name. In any case P' is executed with a channel name associated with v' substituted for x.

4.2.2 The dynamics of filtering

Because of name mobility, the interface between a process and its environment can evolve as they exchange new names in messages. To capture this dynamic aspect of communication, we draw on techniques developed in earlier work on the join-calculus [18]: we use mutually recursive definitions that filter all messages and unfold new filters for their arguments. Some complications arise because the join-calculus that we use here features several types of channels. Accordingly, we maintain a key-channel association table for every type that may traverse the firewall.

Assume that S is a finite set of names and that Σ is a set of channel types that contains all the types of names in S and that is closed under decomposition (that is, if $\langle \sigma_1, \ldots, \sigma_n \rangle \in \Sigma$ then $\sigma_1, \ldots, \sigma_n \in \Sigma$). For every $\sigma \in \Sigma$, let S_σ be the set of names of type σ in S.

The definition $D_\Sigma(S)$ creates an association table for every type σ in Σ, with an entry for each name in S_σ. This definition is given next, top-down.

$$
\begin{array}{rcl}
D_\Sigma(S) & \stackrel{\text{def}}{=} & \bigwedge_{\sigma \in \Sigma} \mathsf{assoc}\ \{(x, x^+) \mid x \in S_\sigma\},\ t_\sigma(x) = T_\sigma,\ t'_\sigma(x^+) = T'_\sigma \\
T_\sigma & \stackrel{\text{def}}{=} & \mathsf{def}\ D_{\sigma, x^\circ \triangleright x} \wedge \mathsf{keys}\ x^+, x^-\ \mathsf{in}\ R_x \mid \mathsf{enter}\ x, x^+ \\
T'_\sigma & \stackrel{\text{def}}{=} & \mathsf{def}\ D_{\sigma, x \triangleright E_x}\ \mathsf{in}\ \mathsf{enter}\ x, x^+ \\
D_{\langle \sigma_1, \ldots, \sigma_n \rangle, x^\circ \triangleright x} & \stackrel{\text{def}}{=} & \begin{array}{l} x^\circ \langle y_1^+, \ldots, y_n^+ \rangle \triangleright \\ \quad \mathsf{let}\ y_1 = t'_{\sigma_1}(y_1^+)\ \mathsf{in}\ \ldots \mathsf{let}\ y_n = t'_{\sigma_n}(y_n^+)\ \mathsf{in} \\ \quad x\langle y_1, \ldots, y_n \rangle \end{array} \\
D_{\langle \sigma_1, \ldots, \sigma_n \rangle, x \triangleright E_x} & \stackrel{\text{def}}{=} & \begin{array}{l} x \langle y_1, \ldots, y_n \rangle \triangleright \\ \quad \mathsf{let}\ y_1^+ = t_{\sigma_1}(y_1)\ \mathsf{in}\ \ldots \mathsf{let}\ y_n^+ = t_{\sigma_n}(y_n)\ \mathsf{in} \\ \quad E_x[y_1^+, \ldots, y_n^+] \end{array}
\end{array}
$$

We may write $D_{x \triangleright E_x}$ and $D_{x^\circ \triangleright x}$ for $D_{\sigma, x \triangleright E_x}$ and $D_{\sigma, x^\circ \triangleright x}$, respectively, when σ is unimportant or clear from context.

In our construction of a firewall in section 4.2.3, we use a definition $D_\Sigma(S)$ for intercepting and forwarding messages, performing conversions between channel names and keys, encrypting and decrypting, and unfolding new filters for processing future messages. We explain $D_\Sigma(S)$ further by distinguishing two ways in which entries appear in association tables at the firewall:

- Suppose that the entry (x, x^+) has been registered in an association table at the firewall from the inside, by a call to $t_\sigma(x)$. This call would have allocated a new pair of keys x^+, x^-, unfolded a definition $D_{\sigma, x^\circ \triangleright x}$, and forked a process R_x.

  Whenever a process outside the firewall knows the key x^+ and produces a message on x by running the process $E_x[y_1^+, \ldots, y_n^+]$ on the public network, the process R_x first performs a decryption with x^-, then delivers the message $x^\circ\langle y_1^+, \ldots, y_n^+\rangle$ to a second stage.

  In this second stage, the effect of receiving $y_1^+, \ldots, y_n^+$ on x° is described in $D_{\sigma, x^\circ \triangleright x}$. According to $D_{\sigma, x^\circ \triangleright x}$, the encryption keys received $(y_1^+, \ldots, y_n^+)$ are mapped to channel names $(y_1, \ldots, y_n)$. For this purpose, the firewall performs a lookup for every key y_i^+ in the association table corresponding to the expected type for y_i. Once every name y_i has been obtained, the plain message $x\langle y_1, \ldots, y_n\rangle$ is delivered to some definition inside the firewall.

  In case a key y_i^+ is not present in the corresponding association table, the allocator T'_{σ_i} registers a fresh name y_i so that future messages sent on y_i inside the firewall are encrypted and forwarded outside using the key y_i^+. Note that, as a result of this method, the same key y_i^+ may eventually appear in the association tables for several different types, and be associated with a different channel name in each of these tables.

- In the other direction, suppose that the entry (x, x^+) has been registered in an association table at the firewall from the outside, by a call to $t'_\sigma(x^+)$. This call would have given rise to a definition $D_{\sigma, x \triangleright E_x}$ that binds x.

  Whenever a message $x\langle y_1, \ldots, y_n\rangle$ is emitted within the firewall, first every channel name y_i is replaced with an encryption key y_i^+ (possibly extending the association tables if these names are crossing the firewall for the first time), then the resulting contents are encrypted by the sending process $E_x[y_1^+, \ldots, y_n^+]$.

The definition of $D_\Sigma(S)$ extends smoothly to variants of the join-calculus with basic types (such as the types of integers and strings). Instead of including an association table in $D_\Sigma(S)$ for each basic type σ, we assume a given bijective function t_σ for marshaling values from σ to $\mathsf{BitString}$ and its inverse t'_σ for unmarshaling values from $\mathsf{BitString}$ to σ.

4.2.3 Putting a process behind a firewall

Next, we show how to construct a firewall for a process. The firewall will depend only on the output interface of the process, that is, on the free names of the process. We bind all those free names in the firewall, and register them associating each of them with a free encryption key. When the process exports other names, as it runs, the firewall evolves too. Thus, we guarantee that the firewall intercepts every message between the process and its environment. We put a process P behind a firewall simply by placing it in a context $\mathcal{F}_S[\,\cdot\,]$, where S includes all the free names of P. The context $\mathcal{F}_S[\,\cdot\,]$ is defined by:

$$\mathcal{F}_S[P] \quad \stackrel{\text{def}}{=} \quad \mathsf{def}\ D_\Sigma(S) \wedge \bigwedge_{x \in S} D_{x \triangleright E_x}\ \mathsf{in}\ P$$

where Σ is the smallest set of types that contains the types of all the names in S and that is closed under decomposition. Every well-typed join-calculus process uses only a finite number of types [17], so Σ is finite, so the context $\mathcal{F}_S[\,\cdot\,]$ is well-defined. Without loss of generality, we can assume that the lookup functions t_σ, t'_σ are not free in P.

For each name $x \in S$ of type σ, the firewall of $\mathcal{F}_S[P]$ has an entry (x, x^+) in the table for σ and a definition $D_{x \triangleright E_x}$. This definition takes outputs from P on x, marshals them, and relays them using encryption with the key x^+. In the course of computation, though, P may export the names of channels defined in P, for input. Accordingly, it is sometimes useful to describe the state of a firewall that exports encryption keys that correspond to those names.

We therefore generalize the definition of the firewall context as follows. Given a process $P \stackrel{\text{def}}{=} \mathsf{def}\ D\ \mathsf{in}\ Q$, assume that S and X are two finite disjoint sets of names such that S includes all the free names of P and X includes only names defined in D. We let:

$$\mathcal{F}_S^X[D,\ Q] \quad \stackrel{\text{def}}{=} \quad \mathsf{def}\ D_\Sigma(X \cup S) \wedge \bigwedge_{x \in X} D_{x^\circ \triangleright x} \wedge \bigwedge_{x \in S} D_{x \triangleright E_x} \wedge D\ \mathsf{in}\ Q \mid \prod_{x \in X} R_x$$

where Σ is the smallest set of types that contains the types of all the names in $X \cup S$ and that is closed under decomposition.

Intuitively, S is the output interface of P while X is its input interface. This input interface consists of names in D that have been exported. For every such name x, the firewall has an entry (x, x^+) in an association table and a definition $D_{x \triangleright E_x}$. In addition, the firewall forks a process R_x for receiving messages encrypted with x^+.

The firewall is well-typed and provides a well-typed interface for the process that it encloses. Its only interface with the environment consists of the names *emit* and *recv* and of values of type $\mathsf{BitString}$. It does not make any further assumptions about the types used by the environment. The environment may even overload one of its keys, using it as though it represented several channels of different types and causing it to appear in several association tables at the firewall, without a security compromise. This property of the firewall is essential: security should not depend on a typing assumption about the environment, since the environment could well violate such an assumption.

The firewall is intended for use in the environment $\mathcal{E}nv\,[\,\cdot\,]$, in particular in the form

$$\mathcal{E}nv\,[\,\mathcal{F}_S[P]\,]$$

The next section relates firewalls to the compositional translation of section 4.1 by establishing the equivalence:

$$\mathcal{E}nv\,[\,\mathcal{F}_S[P]\,] \approx \mathcal{E}nv\,[\,[\![P]\!]\,]$$

where S includes all the free names of P.

4.2.4 An example

We continue the first example (of section 2.1.2), placing the process

$$\mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ x\langle u, v\rangle$$

within a firewall. We write σ and τ for the respective types of the free names u and v, so the free name z has the type $\langle\sigma\rangle$. By putting the process in a firewall context, we obtain:

$$\begin{array}{l}\mathsf{def}\ D_{\{\sigma,\tau,\langle\sigma\rangle\}}(\{u, v, z\}) \wedge D_{u \triangleright E_u} \wedge D_{v \triangleright E_v} \wedge D_{z \triangleright E_z}\ \mathsf{in} \\ \mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ x\langle u, v\rangle\end{array}$$

In this process, each of three association tables contains a single pair ((u, u^+) for type σ; (v, v^+) for type τ; and (z, z^+) for type $\langle\sigma\rangle$), and the firewall has three output filters $D_{u \triangleright E_u}$, $D_{v \triangleright E_v}$, and $D_{z \triangleright E_z}$ for the names u, v, and z. The process can take a local step that releases the message $z\langle u\rangle$. The firewall will intercept this message, fetch the name u^+ from the table for type σ, and run the process $E_z[u^+]$. In three reduction steps, we obtain:

$$\begin{array}{l}\mathsf{def}\ D_{\{\sigma,\tau,\langle\sigma\rangle\}} \wedge D_{u \triangleright E_u} \wedge D_{v \triangleright E_v} \wedge D_{z \triangleright E_z}\ \mathsf{in} \\ E_z[u^+] \mid \mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ 0\end{array}$$

This process will behave like $E_z[u^+]$, because both the firewall and the process $\mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ 0$ are inert.

5 Soundness results

In this section, we present our main soundness results. First, we define the relation of equivalence on processes, mentioned informally in other sections. Then, we state our main theorems and discuss their significance.

Appendix A contains additional formal details. It contains, in particular, scoping rules and operational semantics, including the definition of binary relations $\equiv$ and $\rightarrow$ on processes. Intuitively, $P \equiv Q$ means that P and Q are structurally equivalent (almost syntactically equivalent, modulo some rearrangements); and $P \rightarrow Q$ means that P may perform one step of internal computation and then behave as Q. Appendix B contains auxiliary results and proofs.

5.1 Observational equivalence and other relations

We use a form of observational equivalence as a basis for reasoning about processes. The concept of observational equivalence goes back to Milner's work on CCS [35]; more specifically, we adopt a reduction-based semantics, as for example in the work of Honda and Yoshida [24] (see also [19]). Next, we define observational equivalence and other relations; all the definitions are fairly standard.

Evaluation contexts are the contexts generated by the following grammar:

$$C[\cdot] \quad ::= \quad [\cdot] \;\Big|\; P|C[\cdot] \;\Big|\; \mathsf{def}\ D \ \mathsf{in}\ C[\cdot]$$

where P ranges over processes and D over definitions of the calculus under consideration (here, the join-calculus or the sjoin-calculus). Every evaluation context is structurally equivalent to one of the form $P \mid [\cdot]$ or $\mathsf{def}\ D\ \mathsf{in}\ P \mid [\cdot]$. Further, P can be restricted to the form $P_1 \mid \ldots \mid P_n \mid [\cdot]$, for some $n \geq 0$, where none of $P_1, \ldots, P_n$ is a definition or a parallel composition. In analyzing evaluation contexts, we often consider only contexts of the form $\mathsf{def}\ D\ \mathsf{in}\ P \mid [\cdot]$; the case $P \mid [\cdot]$ is analogous but simpler.

For every process P and name x, we define the assertions $P \downarrow_x$ ("P has strong barb x") and $P \Downarrow_x$ ("P has barb x") as follows:

$$\begin{array}{rcl} P \downarrow_x & \stackrel{\text{def}}{=} & \exists C[\cdot], \widetilde{v}.\, P \equiv C[x\langle\widetilde{v}\rangle] \\ & & \quad \textit{with } C[\cdot] \textit{ an evaluation context that does not bind } x \\ P \Downarrow_x & \stackrel{\text{def}}{=} & \exists P'.\, P \rightarrow^* P' \wedge P' \downarrow_x \end{array}$$

where $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$. That is, $P \downarrow_x$ holds if P may output on x immediately, and $P \Downarrow_x$ holds if P may output on x either immediately or after some internal reductions.

Observational equivalence ($\approx$) is the largest symmetric relation $\mathcal{R}$ on processes that meets the following requirements:

1. if $P \mathrel{\mathcal{R}} Q$ and $P \Downarrow_x$ then $Q \Downarrow_x$, for every name x;
2. $\mathcal{R}$ is closed by application of evaluation contexts, that is, for all evaluation contexts $C[\cdot]$, if $P \mathrel{\mathcal{R}} Q$ then $C[P] \mathrel{\mathcal{R}} C[Q]$;
3. $\mathcal{R}$ is a weak bisimulation, that is, for all P and Q we have:

$$P \mathrel{\mathcal{R}} Q \text{ and } P \rightarrow P' \text{ implies } \exists Q'.\, Q \rightarrow^{*} Q' \text{ and } P' \mathrel{\mathcal{R}} Q'$$

May-testing equivalence ($\simeq_{may}$) is the largest symmetric relation that meets only requirements 1 and 2. May-testing equivalence is coarser than observational equivalence in that it does not take into account the branching structure of processes.

We also make use of an auxiliary *expansion relation* [41] which is finer than observational equivalence: *barbed expansion* ($\succeq$) is the largest relation $\mathcal{R}$ such that $\mathcal{R}$ and its converse $\mathcal{R}^{-1}$ meet requirements 1, 2, 3, and such that $\mathcal{R}$ meets the asymmetric requirement:

$$P \mathrel{\mathcal{R}} Q \text{ and } P \rightarrow P' \text{ implies } \exists Q'.\, Q \rightarrow^{=} Q' \text{ and } P' \mathrel{\mathcal{R}} Q'$$

where $\rightarrow^{=}$ is the reflexive closure of $\rightarrow$. We write $\preceq$ for the converse of $\succeq$.

The definitions above are parametric in the reduction relation $\rightarrow$ and in the grammar of evaluation contexts. Here, they define relations for both the join-calculus and the sjoin-calculus. We refer to earlier papers (e.g., [18, 19, 17]) for further discussion of these definitions.

5.2 Correctness of the low-level protocols for communication

We first define a correctness property for protocols that transmit a single message, such as those of section 3. Our translations are based on such protocols, and this correctness property suffices as a hypothesis in results about the translations. (An even weaker property may suffice, but its definition would probably be fairly complex.) Compared to common specifications of cryptographic protocols, our correctness property is rather abstract: it concerns the intended effects of a protocol rather than the specifics of its implementation.

In the following definition, a protocol is a pair $(R_x,\, E_x[\cdot])$ consisting of a process for receiving and one for sending, parameterized by a channel name x. The definition relies on a set $\mathbf{R}$ of derivatives of the receiving process R_x. This set represents the different states of a receiver after interaction with its context. The definition also relies on the context

$$\mathcal{E}nv_{x^{\pm}}\,[\,\cdot\,] \quad \stackrel{\text{def}}{=} \quad \mathcal{E}nv\,\big[\,\mathsf{def\ keys}\ x^{+}, x^{-}\ \mathsf{in}\ \mathit{plug}'\langle x^{+}\rangle \mid [\cdot]\,\big]$$

This context publishes the encryption key x^{+} through the channel plug', but does not publish the decryption key x^{-}. Although $\mathcal{E}nv_{x^{\pm}}\,[\,\cdot\,]$ does not define x°, we usually nest it in other contexts that define x° in appropriate ways.

Definition 1 *The protocol $(R_x, E_x[\,\cdot\,])$ is correct if there is a set of processes $\mathbf{R}$ that satisfies the following conditions.*

1. *$R_x \in \mathbf{R}$.*

2. *The free names of $E_x[\,\cdot\,]$ are at most emit, recv, and names of type $\mathsf{BitString}$. For every $R \in \mathbf{R}$, the free names of R are at most emit, recv, x°, and names of type $\mathsf{BitString}$.*

3. *For every $R \in \mathbf{R}$, it is not the case that $R \downarrow_{emit}$ or that $R \downarrow_{x^\circ}$.*

4. *For every $R \in \mathbf{R}$, for every tuple $\widetilde{v}$ of values of type $\mathsf{BitString}$ whose length matches the arity of x, if x^- does not occur in $\widetilde{v}$, then*

$$\mathcal{E}nv_{x^\pm}\,[\,R \mid E_x[\widetilde{v}]\,] \;\succeq\; \mathcal{E}nv_{x^\pm}\,[\,R \mid x^\circ\langle\widetilde{v}\rangle\,]$$

5. *For every value v, if x^+ and x^- do not occur in v and*

$$\mathcal{E}nv_{x^\pm}\,[\,R_x \mid emit\langle v\rangle\,] \rightarrow P$$

then

$$P \succeq \mathcal{E}nv_{x^\pm}\,[\,R_x \mid emit\langle v\rangle\,]$$

6. *For every $R \in \mathbf{R}$, if x^- does not occur in v and*

$$\mathcal{E}nv_{x^\pm}\,[\,R \mid emit\langle v\rangle\,] \rightarrow P$$

then

$$P \succeq \mathcal{E}nv_{x^\pm}\,[\,R' \mid Q\,]$$

for some $R' \in \mathbf{R}$ and some process Q such that x^- does not occur in Q.

Condition 1 says that the initial receiving process, R_x, is in $\mathbf{R}$.

Condition 2 restricts the free names in use in the protocol. The sending process $E_x[\,\cdot\,]$ may have access to the network-interface channels (*emit* and *recv*). The receiving process R_x and all other processes in $\mathbf{R}$ may have access to those channels and also to x°. The requirement that x° does not occur free in $E_x[\,\cdot\,]$ rules out degenerate protocols where the sending process does the work of the receiving process, like $(\mathsf{0}, x^\circ\langle\cdot\rangle)$. In addition to *emit*, *recv*, and x°, the protocol may rely on names of type $\mathsf{BitString}$. Intuitively, condition 2 implies that communication from $E_x[\,\cdot\,]$ to R_x is limited to messages of types $\mathsf{BitString}$ and $\langle\mathsf{BitString}\rangle$ on the channels *emit* and *recv*. Therefore, the protocol can be directly implemented over an ordinary network like that represented by the channels *emit* and *recv*, without any additional assumptions about physical security or out-of-band communication.

Condition 3 says that every process $R \in \mathbf{R}$ is passive, in the sense that it never sends messages on *emit* or x° on its own. This condition still allows R to send messages on *recv*.

Condition 4 says that the protocol transmits messages reliably and secretly when an instance of the sending process $E_x[\cdot]$ is put in parallel with the receiving process R_x or any other process $R \in \mathbf{R}$. Using the expansion relation, this condition compares $R \mid E_x[\widetilde{v}]$ with $R \mid x^\circ\langle\widetilde{v}\rangle$. The former process has $E_x[\widetilde{v}]$ as a component, while in the latter $E_x[\widetilde{v}]$ is replaced with its intended outcome, namely $x^\circ\langle\widetilde{v}\rangle$ (with no other visible effect). In both processes we have the same component R; this condition implies that the state of the receiving process remains essentially unchanged as long as it interacts with regular sending processes. Condition 4 rules out insecure protocols that leak information in the course of transmitting a message; many obviously insecure protocols fall into this category. In particular, the naive protocol of section 3 violates condition 4 on several counts. For example, a context that intercepts the message from $E_x[\widetilde{v}]$ and listens on x° can differentiate $\mathcal{E}nv_{x^\pm}[R \mid E_x[\widetilde{v}]]$ and $\mathcal{E}nv_{x^\pm}[R \mid x^\circ\langle\widetilde{v}\rangle]$. Furthermore, a context that interacts with $\mathcal{E}nv_{x^\pm}[R \mid E_x[\widetilde{v}]]$ may be able to guess $\widetilde{v}$, then confirm the guess by computing $\{\widetilde{v}\}_{x^+}$ and comparing $\{\widetilde{v}\}_{x^+}$ with the message from $E_x[\widetilde{v}]$. In contrast, the context cannot obtain the same information in interaction with $\mathcal{E}nv_{x^\pm}[R \mid x^\circ\langle\widetilde{v}\rangle]$.

The last two conditions (5 and 6) describe interactions between the receiving process and the context. These conditions are needed in addition to condition 4 because the context might not behave as a regular sending process. Since the context may communicate with the receiving process only through messages of the form $emit\langle v\rangle$, the last two conditions describe the behavior of the receiving process in reaction to such a message. The conditions concern two cases, distinguished by whether the encryption key x^+ occurs in v.

Condition 5 describes the behavior of R_x in a context that does not have access to the keys x^+ or x^-. Intuitively, this behavior is that exhibited when the attacker has not yet been given x^+, so the receiving process should remain essentially invisible. For every reduction $\mathcal{E}nv_{x^\pm}[R \mid emit\langle v\rangle] \to P$, expansion relates the outcome P to the initial state $\mathcal{E}nv_{x^\pm}[R \mid emit\langle v\rangle]$. Thus, if R_x takes a message from the network and the message is unrelated to x^+, x^-, then R_x must resend the message. Similarly, if R_x becomes a process R by internal reductions (so P equals $\mathcal{E}nv_{x^\pm}[R \mid emit\langle v\rangle]$), then it must be possible to go back from R to R_x obtaining that $\mathcal{E}nv_{x^\pm}[R \mid emit\langle v\rangle] \succeq \mathcal{E}nv_{x^\pm}[R_x \mid emit\langle v\rangle]$.

For example, condition 5 excludes a protocol where R_x "swallows" all messages that are not encrypted under x^+, thus revealing its presence. It also excludes a protocol where R_x "swallows" messages selectively, possibly revealing sensitive information.

Condition 6 describes the behavior of a receiving process $R \in \mathbf{R}$ in a context that has access to the encryption key x^+ but not to the decryption key x^-. Intuitively, this behavior corresponds to the case where the attacker has been given the encryption key, and can thus cause messages on x°; in this situation, the attacker should still not interfere with messages from other senders. The reduction $\mathcal{E}nv_{x^\pm}\,[\,R \mid emit\langle v\rangle\,] \rightarrow P$ may change the state of the receiving process, for example by completing a run of the protocol and relaying a message on x°. The process P cannot however be arbitrary: it must be in the expansion relation with a process that includes a new receiving process $R' \in \mathbf{R}$ in place of R, in parallel with a process Q in place of $emit\langle v\rangle$. The process Q may contain emissions on x° and $emit$; it typically consists of parts of residues of R that do not need the decryption key x^- any more.

For example, condition 6 excludes an insecure variant of a correct protocol where the receiving process is modified as follows. In answer to messages of a form that a regular sending process never creates (for example, $\{c, c, c\}_{x^+}$ for the basic protocol), the receiving process emits a fresh value u; the receiving process works correctly for input tuples that do not contain u, but leaks input tuples that contain u. A flaw appears only after the creation of u, which regular sending processes never cause. Thus, all other conditions can be met. Nonetheless, an attacker may obtain u, pass u to some third party, and harvest secrets later on if the third party includes u in messages to the receiving process. This variant violates condition 6: consider the reduction $\mathcal{E}nv_{x^\pm}\,[\,R \mid emit\langle\{\widetilde{v}\}_{x^+}\rangle\,] \rightarrow P$ that will cause the creation of u; then condition 6 requires that $P \succeq \mathcal{E}nv_{x^\pm}\,[\,R' \mid Q\,]$ for some $R' \in \mathbf{R}$, while no such R' can satisfy all the other conditions.

Theorem 1 *Both the basic protocol and the session-based protocol are correct.*

The proof of this theorem, which is presented in Appendix B, relies crucially on the use of noise in our definition of the network. For the basic protocol, the set $\mathbf{R}$ is indexed by the content of the table of unique identifiers. For the session-based protocol, the set $\mathbf{R}$ is simply the singleton $\{R_x\}$.

For the rest of this section, we consider an arbitrary but fixed correct protocol. In particular, whenever we refer to a translation, we mean a translation built on this protocol.

5.3 Main results

Intuitively, we may think of $[\![P]\!]$ and $\mathcal{F}_S[P]$ as implementations of P. Technically, this intuition is not quite satisfactory, particularly if we adopt a standard concept of implementation according to which Q implements P if and only if every possible behavior of Q is a possible behavior of P (e.g., [25]).

1. This intuition is not accurate, because $[\![P]\!]$ and $\mathcal{F}_S[P]$ may behave in ways in which P could not behave. In particular, $[\![P]\!]$ and $\mathcal{F}_S[P]$ typically send messages on the channels *emit* and *recv* even when these names are not present in P.

2. This intuition is not quite sufficient, because it does not imply that the proposed implementation shares the security properties of P: the implementation relation need not preserve those properties (e.g., [31, 32]).

Security properties can sometimes be phrased as equivalences (as in section 2.1), so the second objection suggests that we phrase our result in terms of equivalences rather than in terms of implementation relations. The first objection suggests that we should take the context $\mathcal{E}nv\,[\,\cdot\,]$ into account and that we should not directly equate P with $[\![P]\!]$ and $\mathcal{F}_S[P]$.

With these insights, we arrive at the following theorem about the compositional translation of section 4.1. In programming language terminology, this theorem is a full-abstraction result. From a security perspective, it implies that low-level attackers in the sjoin-calculus have the same power as higher-level attackers in the join-calculus (see section 5.4).

Theorem 2 *The compositional translation is fully-abstract, up to observational equivalence: for all join-calculus processes P and Q,*

$$P \approx Q \quad \textit{if and only if} \quad \mathcal{E}nv\,[\,[\![P]\!]\,] \approx \mathcal{E}nv\,[\,[\![Q]\!]\,]$$

The statement of this theorem may seem ambiguous, because $P \approx Q$ can be read as the equivalence of P and Q either as join-calculus processes or as sjoin-calculus processes. However, these two readings coincide (see Appendix A.3).

An extreme special case is that where a process P never communicates with its context, so $P \approx \mathsf{0}$. In this case we obtain $\mathcal{E}nv\,[\,[\![P]\!]\,] \approx \mathcal{E}nv\,[\,[\![\mathsf{0}]\!]\,]$, that is, $\mathcal{E}nv\,[\,[\![P]\!]\,] \approx \mathcal{E}nv\,[\,\mathsf{0}\,]$. We may infer (informally) that a low-level attacker cannot tell whether the implementation of P is running. Another simple example is that of the processes $\mathsf{def}\ x\langle y, y'\rangle \triangleright z\langle y\rangle\ \mathsf{in}\ x\langle u, v\rangle$ and $z\langle u\rangle$, as discussed in sections 2.1.2 and 4.1.2.

The next theorem says that the filtering translation of section 4.2 is equivalent to the compositional translation.

Theorem 3 *The filtering translation is equivalent to the compositional translation, up to observational equivalence: for every join-calculus process P and every set of names S that includes all the free names of P,*

$$\mathcal{E}nv\,[\,[\![P]\!]\,] \quad \approx \quad \mathcal{E}nv\,[\,\mathcal{F}_S[P]\,]$$

This theorem implies that the filtering translation is fully abstract too. In this case, however, the "only if" direction of full abstraction is rather trivial, as it follows from basic congruence properties.

In general, full-abstraction results with respect to a particular equivalence relation do not imply full-abstraction results for coarser equivalences. However, we have full-abstraction results for a large class of equivalences.

Theorem 4 *Let $\mathcal{R}$ be a transitive relation on sjoin-calculus processes that is closed by application of evaluation contexts and such that $\approx\ \subseteq \mathcal{R}$. For all join-calculus processes P and Q,*

$$P \mathrel{\mathcal{R}} Q \quad \textit{if and only if} \quad \mathcal{E}nv\,[\,[\![P]\!]\,] \mathrel{\mathcal{R}} \mathcal{E}nv\,[\,[\![Q]\!]\,]$$

In particular, may-testing equivalence ($\simeq_{may}$) is coarser than observational equivalence and is closed by application of evaluation contexts. (It is perhaps the most important relation that satisfies these conditions, because of its correspondence with safety properties [15].) Therefore, we obtain, for example:

$$P \simeq_{may} Q \quad \text{if and only if} \quad \mathcal{E}nv\,[\,[\![P]\!]\,] \simeq_{may} \mathcal{E}nv\,[\,[\![Q]\!]\,]$$

Theorem 3 immediately implies that analogous results hold for the filtering translation.

5.4 Discussion

The "upwards" direction of our full-abstraction results says that two join-calculus processes are equivalent only if their implementations in the sjoin-calculus are equivalent. This property is important because it means that our translations do not identify "too many" join-calculus processes or even—trivially—all join-calculus processes.

For security, however, full abstraction is interesting mainly in the "downwards" direction [1]: if two join-calculus processes are equivalent, then their implementations in the sjoin-calculus are equivalent too. Hence, it is sound to reason about a join-calculus process and to infer properties of its sjoin-calculus implementation, without explicitly considering the low-level communication mechanisms of the implementation and without concern for low-level attacks on those communication mechanisms.

For example, suppose that we have written a join-calculus process that publishes a communication channel (like the contest server of section 2.1.3 publishes *entry*). We can assess the consequences of this publication within the high-level calculus, considering only the high-level contexts in which the process may be used. Those consequences may be undesirable—our results do not exclude that possibility. Our results do imply that we need not think about lower-level details of how the channel is represented by keys, or about low-level attacks that this representation might permit.

More precisely, suppose that x is a channel name free in P, that y is bound in P, and that P includes y in a message on x. The implementation of P will relay a key y^+ instead of y to the outside. Perhaps y was intended for internal use only, so messages on y may be trusted in P somehow (wrongly). Then an attack against the implementation may be able to exploit the knowledge of y^+, since the implementation will decrypt any message encrypted with y^+ and relay its contents internally. In this situation, our techniques do not help, but they do not hurt either (as Theorem 2 indicates). The translations could be supplemented with restrictions for auditing or preventing the escape of keys like y^+, at the cost of some transparency.

We should mention, as a caveat, that the low-level attacks that we can rule out are only those that can be expressed in the sjoin-calculus. Although the sjoin-calculus is rather rich, it does have some limitations. For example, the sjoin-calculus represents key generation and encryption rather abstractly, not showing the details of particular algorithms for these purposes; therefore, the shortcomings or peculiarities of those algorithms are beyond the scope of our results. Moreover, the join-calculus and the sjoin-calculus do not include any notion of real time or of network bandwidth; therefore, they exclude from consideration any attack that depends on observing real-time behavior or bandwidth usage.

Our full-abstraction results are rather robust, as they hold for a range of equivalence relations on processes, including observational equivalence and may-testing equivalence, and also intermediate equivalences that incorporate notions of fairness, such as fair-testing equivalence [11, 39]. Much as in the spi-calculus, observational equivalence can often be proved more directly than may-testing equivalence. While may-testing equivalence probably has a more intuitive meaning, our results for observational equivalence demonstrate that our translations preserve even the branching structure of processes.

The strength of our results has a price: in order to obtain these results, we have adopted a sophisticated and somewhat expensive implementation strategy. For example, our results would not hold if we did not protect against traffic-analysis attacks. For this protection, we rely on the use of noise. Since in practice noise consumes network bandwidth, it would be attractive to omit the noise; one would still hope to prove a correctness result—at least a weak one. Formulating and proving such results is an intriguing subject for future research.

6 Applications

Since the compositional translation is (obviously) compositional, Theorem 3 implies that the filtering translation is compositional too. Building on this observation, we address issues of distribution by analyzing systems with multiple firewalls. We also illustrate the use of our results in an example.

6.1 Multiple firewalls and their placement

The join-calculus can be refined to account for computations distributed over several sites [20]. For our purposes, distribution simply means a fairly arbitrary partition of processes and definitions. Thus, when we write:

$$\mathsf{def}\ D\ \mathsf{in}\ P \mid Q$$

we may intend that the different parts of this expression (D, P, Q) be executed at several sites interconnected by a public network—although we do not rely on any special syntax to indicate this in the join-calculus. This section shows that, following our method, we can wrap a firewall around the code for each site, making the site boundaries explicit. Each firewall communicates with its outside (including the firewalls for other sites) through a poor interface: the public network, where traffic is unprotected and limited to type $\mathsf{BitString}$. Moreover, we do not assume a particular application protocol for D, P, and Q. Nevertheless, the distribution across sites preserves security properties.

Section 4.2.3 shows how to enclose a single process within a firewall. More generally, firewalls can be placed around any part of a composite system. For example, we can write:

$$\mathcal{E}nv\,[\,\mathcal{F}_S[P] \mid \mathcal{F}_S[Q]\,]$$

representing two separate processes P and Q, each with its own firewall, on a public network modeled as the environment $\mathcal{E}nv\,[\,\cdot\,]$. Each firewall maintains its own association tables; the two firewalls may map an encryption key to different local names of channels. We have the following compositionality property:

Theorem 5 *For all join-calculus processes P and Q, and every finite set of names S that includes all the free names of P and Q,*

$$\mathcal{E}nv\,[\,\mathcal{F}_S[P \mid Q]\,] \quad \approx \quad \mathcal{E}nv\,[\,\mathcal{F}_S[P] \mid \mathcal{F}_S[Q]\,]$$

This property means that we can partition the process $P \mid Q$ into two components P and Q, locate these components at different sites with their own firewalls, let them communicate with each other and with the environment through the firewalls, and be sure that the environment will not be able to detect any difference with the situation where $P \mid Q$ is at a single site with one firewall.

Another compositionality property deals with the distribution of both processes and definitions over any number of sites; it is analogous but more complex and more general. In the statement of this property, we rely on the following notation, which extends the notation $\mathcal{E}nv_{x^\pm}[\,\cdot\,]$ replacing x with a finite set of names X:

$$\mathcal{E}nv_{X^\pm}[\,\cdot\,] \quad \stackrel{\text{def}}{=} \quad \mathcal{E}nv\left[\mathsf{def}\ \bigwedge_{x\in X} \mathsf{keys}\ x^+, x^-\ \mathsf{in}\ [\,\cdot\,]\right]$$

Theorem 6 *Let S be a finite set of names and let P be a join-calculus process of the form $\mathsf{def}\ \bigwedge_{i=1}^n D_i\ \mathsf{in}\ \prod_{i=1}^n Q_i$, where S includes all the free names of P and where the sets $\mathsf{dv}[D_i]$ and S are pairwise disjoint. For $i, j \leq n$, we define the sets of names $X_i^j = \mathsf{dv}[D_i] \cap (\mathsf{fv}[D_j] \cup \mathsf{fv}[Q_j])$, $X_i = \bigcup_{j\neq i} X_i^j$, $X^j = \bigcup_{i\neq j} X_i^j$, and $X = \biguplus_{i=1}^n X_i$. We have the equivalence*

$$\mathcal{E}nv\,[\,\mathcal{F}_S[P]\,] \quad \approx \quad \mathcal{E}nv_{X^\pm}\left[\prod_{i=1}^n \mathcal{F}_{X^i\cup S}^{X_i}[D_i,\ Q_i]\right]$$

This property means that we can partition the process P into n components and still guarantee that communication on channels defined in P proceeds as though all the components were at the same site. Each component consists of a definition D_i and a process Q_i behind a firewall. Communication between D_i and Q_i is local; it does not involve any firewall. The names in X_i^j represent the input interface of component i made available to component j. Therefore, the names in X_i represent the input interface of component i, and the names in $X^i \uplus S$ represent the output interface of component i. Finally, the set X contains all the names that are local to P but shared between at least two components. For each name $x \in X$, the context $\mathcal{E}nv_{X^\pm}[\,\cdot\,]$ binds a key pair x^+, x^-. The firewalls for the n sites can communicate with one another using x^+, x^- but neither x^+ nor x^- is available to the outside initially.

When Q_i sends a message on a channel $x \in X_j^i$, the message traverses the ith and jth firewalls, undergoing first encryption with x^+ and then decryption with x^-. If the message contains another name $y \in X_i^j \cup X_j^i \cup S$, this name is converted to a key y^+ and back to a channel name as it traverses the firewalls. If the message contains a name z that Q_i is exporting for the first time (for example, $z \in \mathsf{dv}[D_i] \setminus X_i$), a new key z^+ appears and is successively recorded at both firewalls; this key is associated with z at the ith firewall and with a new name at the jth firewall. The result is the establishment of a secure communication path in the reverse direction, from component j back to component i. As other channel names are exchanged, the components keep communicating just as they would do if they were at the same site. The firewalls are transparent, and they extend transparently whenever a new channel name traverses them.

Intuitively, there is no requirement that all the sites trust one another, and in fact some of them may represent incompetent or malicious participants, possibly collaborating with an outside attacker. Our results apply to this situation without change.

6.2 An example

In order to illustrate the use of our method, we revisit the example of section 2.1.3 and describe two implementations of the contest server defined there.

First, we discuss the effect of wrapping a firewall around the contest server in its initial state, which is described by the process:

$$C[\mathit{open}\langle\rangle \mid \mathit{plug}\langle \mathit{entry}\rangle]$$

This process has a single free name, *plug*, but its type is quite complicated. We explain this type bottom-up, writing String for the basic type of strings. The type of a and *closed* is $\langle\mathsf{String}\rangle$, so the type of *entry* is $\langle\langle\mathsf{String}\rangle, \mathsf{String}\rangle$ and the type of *plug* is $\langle\langle\langle\mathsf{String}\rangle, \mathsf{String}\rangle\rangle$. In order to protect the contest server, our firewall contains a table for each of the three types in the set:

$$\Sigma \stackrel{\text{def}}{=} \{\langle\mathsf{String}\rangle, \langle\langle\mathsf{String}\rangle, \mathsf{String}\rangle, \langle\langle\langle\mathsf{String}\rangle, \mathsf{String}\rangle\rangle\}$$

An implementation of the contest server with a single site is described by:

$$I \stackrel{\text{def}}{=} \mathcal{E}nv\left[\mathcal{F}_{\{\mathit{plug}\}}[C[\mathit{open}\langle\rangle \mid \mathit{plug}\langle\mathit{entry}\rangle]]\right]$$

Initially the firewall contains only the pair $(\mathit{plug}, \mathit{plug}^+)$ in the table for the type $\langle\langle\langle\mathsf{String}\rangle, \mathsf{String}\rangle\rangle$. As a result, the message $\mathit{plug}\langle\mathit{entry}\rangle$ is intercepted, the content *entry* is marshaled, a fresh pair of keys $\mathit{entry}^+, \mathit{entry}^-$ is stored in the table for $\langle\langle\mathsf{String}\rangle, \mathsf{String}\rangle$, and two processes are forked: $E_{\mathit{plug}}[\mathit{entry}^+]$ which encrypts and sends entry^+, and R_{entry} which awaits messages encrypted with entry^+. The evolution of I is summarized in the following equivalence:

$$I \approx \mathcal{E}nv\left[\mathsf{def\ keys}\ \mathit{entry}^+, \mathit{entry}^-\ \mathsf{in}\ E_{\mathit{plug}}[\mathit{entry}^+] \mid \mathcal{F}^{\{\mathit{entry}\}}_{\{\mathit{plug}\}}[D,\ \mathit{open}\langle\rangle]\right]$$

where D is the definition such that $C[\,\cdot\,] = \mathsf{def}\ D\ \mathsf{in}\ [\,\cdot\,]$. The environment can then obtain the encryption key entry^+, typically but not necessarily by using a process R_{plug}, and hence can participate in the contest. Later, whenever a new entry is received by R_{entry} and unmarshaled, it provides a key for encrypting a response; a local continuation channel of type $\langle\mathsf{String}\rangle$ is allocated for this response. The size of the table for $\langle\mathsf{String}\rangle$ may grow each time an entry is processed.

Similarly, we can wrap three-table firewalls around all the processes of section 2.1.3. By applying Theorem 2 to the security properties stated in section 2.1.3, we obtain corresponding implementation-level security properties.

As a second implementation of the contest server, we consider the variant with proxies. This variant can naturally be mapped to several sites, each protected by a firewall of the kind described above. Applying Theorem 6, we obtain the following equivalence, which expresses that the distribution of proxies is transparent:

$$
\begin{array}{rl}
& \mathcal{E}nv\left[\mathcal{F}_{\{plug_1, plug_2\}}\left[\begin{array}{rc} & open\langle\rangle \\ \mathsf{def}\ D\ \mathsf{in}\ | & C_1[open_1\langle\rangle \mid plug_1\langle entry_1\rangle] \\ | & C_2[open_2\langle\rangle \mid plug_2\langle entry_2\rangle] \end{array}\right]\right] \\
\approx & \mathcal{E}nv\left[\begin{array}{l} \mathsf{def\ keys}\ entry^+, entry^-\ \mathsf{in} \\ \quad \mathcal{F}_\emptyset^{\{entry\}}[D,\ open\langle\rangle] \\ \mid \quad \mathcal{F}_{\{plug_1, entry\}}[C_1[open_1\langle\rangle \mid plug_1\langle entry_1\rangle]] \\ \mid \quad \mathcal{F}_{\{plug_2, entry\}}[C_2[open_2\langle\rangle \mid plug_2\langle entry_2\rangle]] \end{array}\right]
\end{array}
$$

The top process describes a centralized system where the contest server and the proxies are all at a single site protected by a firewall. The bottom process describes a distributed implementation with three components protected by their own firewalls. (Theorem 6 yields additional, superfluous occurrences of $plug_1$ and $plug_2$ in the three firewalls; we have removed those occurrences using simple equivalences.) The two processes have different structures, but participants in the contest cannot distinguish them. More importantly, perhaps, attackers cannot distinguish these two processes either—so communication between the proxies and the server over the public network does not enable attacks.

7 Conclusions

This paper describes an implementation of a high-level language with secure channels in terms of a lower-level language with primitives for encryption and decryption. The implementation relies on cryptographic protocols for protecting communication over a public network. The design and analysis of good cryptographic protocols always requires both ingenuity and care. However, this paper does not focus on cryptographic protocols, but instead takes a broader perspective. It considers how the protocols fit into larger systems, and proves that we can reason about programs in the high-level language without explicit concern for lower-level attacks on the protocols.

Several variants and extensions of our results would be attractive, for example the treatment of remote method invocation in a typed object-oriented language, and the mapping down to a low-level language where keys are not formal symbols but finite sequences of bits. More generally, many interesting questions in security can be seen as problems of refinement—relating views at different levels of abstraction (e.g., [23]). Different views of security are appropriate for different communities of users, administrators, designers, and programmers. The gaps between these views are sources of confusion and vulnerability. Unfortunately, some of these gaps are still a fuzzy subject for research, because they stem from fuzzy human expectations.

Appendices

Notation Let $\mathcal{R}$ and $\mathcal{R}'$ be two binary relations. We write $\mathcal{R}\mathcal{R}'$ for the composition of $\mathcal{R}$ and $\mathcal{R}'$, when this composition is defined, so $x\ \mathcal{R}\mathcal{R}'\ z$ if and only if there exists y such that $x\ \mathcal{R}\ y$ and $y\ \mathcal{R}\ z$. We write $\mathcal{R}^=$ for the reflexive closure of $\mathcal{R}$, and $\mathcal{R}^*$ for its reflexive and transitive closure. We use $\uplus$ for disjoint set union; that is, $S \uplus S'$ represents the union of the sets S and S' plus the requirement that S and S' be disjoint.

A Semantic preliminaries

In this section, we detail the operational semantics of the sjoin-calculus and discuss the definition of observational equivalence. This section serves also as a review of the semantics of the join-calculus, since it is a special case of that of the sjoin-calculus.

Formally, we treat repl, uids, and assoc as primitive constructs of the sjoin-calculus (but not of the join-calculus). We therefore give the operational semantics of these constructs in this section. The relation of observational equivalence is not affected by whether these constructs are encoded or taken as primitive. On the other hand, taking them as primitive does impact the expansion relation and is convenient for proofs.

Accordingly, we supplement the type system of the sjoin-calculus with a type for the names used in accessing sets of unique identifiers, and, for each join-calculus type σ, with a type for each of the two names used in accessing association tables that relate values of type σ to values of type $\mathsf{BitString}$. We refer to these names as table names.

We extend the notation for sets of unique identifiers in order to represent the state of these sets. The definition $\mathsf{uids}\ t\{\widetilde{u}\}$ denotes a table that contains the values $\widetilde{u}$. Whenever we write $\mathsf{uids}\ t\{\widetilde{u}\}$, the values $\widetilde{u}$ are pairwise distinct. We identify $\mathsf{uids}\ t\{\}$ with $\mathsf{uids}\ t$.

We also introduce some auxiliary notation for association tables. We abbreviate $\mathsf{assoc}\ S,\ t(x) = T,\ t'(x^+) = T'$ to $\mathsf{assoc}\ S, t, t'$ when the elided components are clear from context. Additionally, we simply omit the set S when it is empty.

A.1 Scopes

First, in Figure 3, we define the sets of free names ($\mathsf{fv}[v]$, $\mathsf{fv}[P]$, and $\mathsf{fv}[D]$), defined names ($\mathsf{dv}[J]$ and $\mathsf{dv}[D]$), and received names ($\mathsf{rv}[J]$), for values, processes, join-patterns, and definitions.

A name is fresh with respect to an expression or set of expressions when it does not occur free in them. We write $\{^{v}/_{x}\}$ for the substitution of the value v for the name x, write $\{^{\widetilde{v}}/_{\widetilde{x}}\}$ for the substitution of the values $v_1, \ldots, v_n$ for the

$$
\begin{array}{rcl}
\mathsf{fv}[x] & \stackrel{\text{def}}{=} & \{x\} \\
\mathsf{fv}[\{v_1,\ldots,v_n\}_v] & \stackrel{\text{def}}{=} & \mathsf{fv}[v] \cup \bigcup_{i\in 1..n} \mathsf{fv}[v_i] \\
\\
\mathsf{fv}[x\langle v_1,\ldots,v_n\rangle] & \stackrel{\text{def}}{=} & \{x\} \cup \bigcup_{i\in 1..n} \mathsf{fv}[v_i] \\
\mathsf{fv}[\mathsf{if}\ v = v'\ \mathsf{then}\ P\ \mathsf{else}\ P'] & \stackrel{\text{def}}{=} & \mathsf{fv}[P] \cup \mathsf{fv}[P'] \cup \mathsf{fv}[v] \cup \mathsf{fv}[v'] \\
\mathsf{fv}[\mathsf{decrypt}\ v\ \mathsf{using}\ v'\ \mathsf{to}\ \widetilde{x}\ \mathsf{in}\ P\ \mathsf{else}\ P'] & \stackrel{\text{def}}{=} & (\mathsf{fv}[P] \setminus \{\widetilde{x}\}) \cup \mathsf{fv}[P'] \cup \mathsf{fv}[v] \cup \mathsf{fv}[v'] \\
\mathsf{fv}[\mathsf{def}\ D\ \mathsf{in}\ P] & \stackrel{\text{def}}{=} & (\mathsf{fv}[P] \cup \mathsf{fv}[D]) \setminus \mathsf{dv}[D] \\
\mathsf{fv}[P \mid P'] & \stackrel{\text{def}}{=} & \mathsf{fv}[P] \cup \mathsf{fv}[P'] \\
\mathsf{fv}[0] & \stackrel{\text{def}}{=} & \emptyset \\
\mathsf{fv}[\mathsf{repl}\ P] & \stackrel{\text{def}}{=} & \mathsf{fv}[P] \\
\mathsf{fv}[\mathsf{if\ not\ tset}\ t(v)\ \mathsf{then}\ P] & \stackrel{\text{def}}{=} & \mathsf{fv}[P] \cup \mathsf{fv}[v] \cup \{t\} \\
\mathsf{fv}[\mathsf{let}\ y = t(v)\ \mathsf{in}\ P] & \stackrel{\text{def}}{=} & (\mathsf{fv}[P] \setminus \{y\}) \cup \mathsf{fv}[v] \cup \{t\} \\
\mathsf{fv}[\mathsf{enter}\ v, v'] & \stackrel{\text{def}}{=} & \mathsf{fv}[v] \cup \mathsf{fv}[v'] \\
\\
\mathsf{rv}[x\langle y_1,\ldots,y_n\rangle] & \stackrel{\text{def}}{=} & \{y_1,\ldots,y_n\} \\
\mathsf{rv}[J \mid J'] & \stackrel{\text{def}}{=} & \mathsf{rv}[J] \uplus \mathsf{rv}[J'] \\
\\
\mathsf{dv}[x\langle y_1,\ldots,y_n\rangle] & \stackrel{\text{def}}{=} & \{x\} \\
\mathsf{dv}[J \mid J'] & \stackrel{\text{def}}{=} & \mathsf{dv}[J] \uplus \mathsf{dv}[J'] \\
\\
\mathsf{dv}[J \triangleright P] & \stackrel{\text{def}}{=} & \mathsf{dv}[J] \\
\mathsf{dv}[\mathsf{fresh}\ x] & \stackrel{\text{def}}{=} & \{x\} \\
\mathsf{dv}[\mathsf{keys}\ x^+, x^-] & \stackrel{\text{def}}{=} & \{x^+, x^-\} \\
\mathsf{dv}[D \wedge D'] & \stackrel{\text{def}}{=} & \mathsf{dv}[D] \cup \mathsf{dv}[D'] \\
\mathsf{dv}[\mathsf{uids}\ t\{v_1,\ldots,v_n\}] & \stackrel{\text{def}}{=} & \{t\} \\
\mathsf{dv}[\mathsf{assoc}\ S,\ t(x) = P,\ t'(x') = P'] & \stackrel{\text{def}}{=} & \{t, t'\} \\
\\
\mathsf{fv}[J \triangleright P] & \stackrel{\text{def}}{=} & \mathsf{dv}[J] \cup (\mathsf{fv}[P] \setminus \mathsf{rv}[J]) \\
\mathsf{fv}[\mathsf{fresh}\ x] & \stackrel{\text{def}}{=} & \{x\} \\
\mathsf{fv}[\mathsf{keys}\ x^+, x^-] & \stackrel{\text{def}}{=} & \{x^+, x^-\} \\
\mathsf{fv}[D \wedge D'] & \stackrel{\text{def}}{=} & \mathsf{fv}[D] \cup \mathsf{fv}[D'] \\
\mathsf{fv}[\mathsf{uids}\ t\{v_1,\ldots,v_n\}] & \stackrel{\text{def}}{=} & \{t\} \cup \bigcup_{i\in 1..n} \mathsf{fv}[v_i] \\
\mathsf{fv}[\mathsf{assoc}\ S,\ t(x) = P,\ t'(x') = P'] & \stackrel{\text{def}}{=} & \\
\end{array}
$$

$$
(\mathsf{fv}[P] \setminus \{x\}) \cup (\mathsf{fv}[P'] \setminus \{x'\}) \cup \{t, t'\} \cup \bigcup_{(v,v')\in S} (\mathsf{fv}[v] \cup \mathsf{fv}[v'])
$$

Figure 3: Scoping rules for the sjoin-calculus.

names $x_1, \ldots, x_n$ when $\widetilde{v} = v_1, \ldots, v_n$ and $\widetilde{x} = x_1, \ldots, x_n$, and let σ range over arbitrary substitutions. We usually identify expressions up to renaming of bound names, assuming implicit α-conversion in order to avoid name clashes. We require that, in every join-pattern, all names be distinct. We also require that, in each definition, each name be defined either in reaction rules or in at most one clause of any other form. (For example, both $t\langle u\rangle \triangleright P \wedge t\langle v\rangle \triangleright Q$ and $\mathsf{keys}\ x, x$ are legal definitions, but $\mathsf{keys}\ x, y \wedge \mathsf{fresh}\ x$ and $\mathsf{uids}\ t \wedge t\langle u\rangle \triangleright P$ are not.)

A.2 Chemical semantics

We present our operational semantics in the chemical style of Berry and Boudol [8], as a variant of the reflexive chemical abstract machine [18].

The state of a computation is represented by a pair of multisets $(\mathcal{D}, \mathcal{P})$, called a *chemical solution*, and written $\mathcal{D} \vdash \mathcal{P}$, where:

- $\mathcal{P}$ is a multiset of processes, intuitively the processes running;
- $\mathcal{D}$ is a multiset of definitions.

The rules for computation operate on chemical solutions. They form two families, *structural rules* and *reduction rules*.

Structural rules are reversible, and express the syntactic rearrangements of expressions in a solution. We write them in the form $\mathcal{D}_1 \vdash \mathcal{P}_1 \rightleftharpoons \mathcal{D}_2 \vdash \mathcal{P}_2$, where $\rightharpoonup$ represents "heating" and $\leftharpoondown$ represents "cooling". We usually omit the parts of $\mathcal{D}_1$, $\mathcal{P}_1$, $\mathcal{D}_2$, and $\mathcal{P}_2$ that are the same on both sides of $\rightleftharpoons$. With this abbreviation convention, we adopt the following structural rules for the sjoin-calculus:

$$
\begin{array}{lrcl}
\textsc{Str-Null} & \vdash \mathsf{0} & \rightleftharpoons & \vdash \\
\textsc{Str-Par} & \vdash P_1 \mid P_2 & \rightleftharpoons & \vdash P_1, P_2 \\
\textsc{Str-And} & D_1 \wedge D_2 \vdash & \rightleftharpoons & D_1, D_2 \vdash \\
\textsc{Str-Def} & \vdash \mathsf{def}\ D\ \mathsf{in}\ P & \rightleftharpoons & D\sigma_{\mathrm{dv}} \vdash P\sigma_{\mathrm{dv}} \\
\textsc{Str-Repl} & \vdash \mathsf{repl}\ P & \rightleftharpoons & \vdash P \mid \mathsf{repl}\ P
\end{array}
$$

with the side condition for Str-Def that $dom(\sigma_{\mathrm{dv}}) = \mathsf{dv}[D]$ and that $range(\sigma_{\mathrm{dv}})$ consists of fresh and distinct names. The first three rules state that $\mid$ and $\wedge$ are associative and commutative, with unit $\mathsf{0}$ for $\mid$. The rule Str-Def describes the introduction of new names and reaction rules in a solution; its side condition ensures that we follow a static scoping discipline.

As an example, we rewrite the rule Str-Def without our abbreviation convention. It becomes: for every process $\mathsf{def}\ D\ \mathsf{in}\ P$ and multisets $\mathcal{D}$ and $\mathcal{P}$, for every substitution σ_{dv} that maps $\mathsf{dv}[D]$ to pairwise distinct names in $\mathcal{N} \setminus$

$(\mathsf{fv}[\mathcal{P}] \cup \mathsf{fv}[\mathcal{D}] \cup \mathsf{fv}[\mathsf{def}\ D\ \mathsf{in}\ P])$,

$$\mathcal{D} \vdash \mathcal{P} \cup \{\mathsf{def}\ D\ \mathsf{in}\ P\} \rightleftharpoons \mathcal{D} \cup \{D\sigma_{\mathrm{dv}}\} \vdash \mathcal{P} \cup \{P\sigma_{\mathrm{dv}}\}$$

Reduction rules represent proper, basic computation steps. We write them in the form $\mathcal{D}_1 \vdash \mathcal{P}_1 \longrightarrow \mathcal{D}_2 \vdash \mathcal{P}_2$. We adopt the following reduction rules for the sjoin-calculus:

$$\begin{array}{llcl}
\textsc{Red} & J \triangleright P \vdash J\sigma_{\mathrm{rv}} & \longrightarrow & J \triangleright P \vdash P\sigma_{\mathrm{rv}} \\[1ex]
\textsc{Compare} & \vdash \mathsf{if}\ v = v\ \mathsf{then}\ P\ \mathsf{else}\ P' & \longrightarrow & \vdash P \\
 & \vdash \mathsf{if}\ v = w\ \mathsf{then}\ P\ \mathsf{else}\ P' & \longrightarrow & \vdash P'
\end{array}$$

$$\begin{array}{rlcl}
\textsc{Decrypt} & \mathsf{keys}\ x^+, x^- & \vdash & \mathsf{decrypt}\ \{\widetilde{v}\}_{x^+}\ \mathsf{using}\ x^-\ \mathsf{to}\ \widetilde{y}\ \mathsf{in}\ P\ \mathsf{else}\ P' \\
\longrightarrow & \mathsf{keys}\ x^+, x^- & \vdash & P\left\{{}^{\widetilde{v}}/_{\widetilde{y}}\right\} \\[2ex]
 & \mathsf{keys}\ x^+, x^- & \vdash & \mathsf{decrypt}\ w\ \mathsf{using}\ x^-\ \mathsf{to}\ \widetilde{y}\ \mathsf{in}\ P\ \mathsf{else}\ P' \\
\longrightarrow & \mathsf{keys}\ x^+, x^- & \vdash & P'
\end{array}$$

with the following side conditions: in Red, that $dom(\sigma_{\mathrm{rv}}) = \mathsf{rv}[J]$; in the second clause of Compare, that $w \neq v$; in the first clause of Decrypt, that $\widetilde{v}$ and $\widetilde{y}$ are tuples of the same length; in the second clause of Decrypt, that w is not of the form $\{\widetilde{v}\}_{x^+}$ with $\widetilde{v}$ and $\widetilde{y}$ of the same length. The rule Red describes the use of a definition clause to consume messages and to produce a new instance of a guarded process. The two rules Compare concern the comparison of values. The two rules Decrypt concern attempts to decrypt values. Note that a process that attempts to decrypt with a non-key will get stuck, like $\mathsf{decrypt}\ w\ \mathsf{using}\ \{\}_{x^+}\ \mathsf{to}\ \widetilde{y}\ \mathsf{in}\ P\ \mathsf{else}\ P'$ which uses $\{\}_{x^+}$ instead of a key; however, this problem does not affect the processes that we construct.

We give additional reduction rules for the constructs uids and assoc, as follows.

$$\begin{array}{rlcl}
\textsc{Uids} & \mathsf{uids}\ t\{\widetilde{u}\} & \vdash & \mathsf{if\ not\ tset}\ t(v)\ \mathsf{then}\ P \\
\longrightarrow & \mathsf{uids}\ t\{\widetilde{u}, v\} & \vdash & P
\end{array}$$

with the side condition $v \notin \{\widetilde{u}\}$. The rule Uids states that the process $\mathsf{if\ not\ tset}\ t(v)\ \mathsf{then}\ P$ atomically tests whether v is present in the set $\{\widetilde{u}\}$ attached to the name t, and if not then it adds v to this set and starts process P. Since there is no other rule that may reduce the process $\mathsf{if\ not\ tset}\ t(v)\ \mathsf{then}\ P$, this process is inert if $v \in \{\widetilde{u}\}$.

$$\begin{array}{rrcl}
\textsc{Assoc} & \mathsf{assoc}\ \{\widetilde{(u, u')}, (v, v')\}, t, t' & \vdash & \mathsf{let}\ y = t(v)\ \mathsf{in}\ P \\
\longrightarrow & \mathsf{assoc}\ \{\widetilde{(u, u')}, (v, v')\}, t, t' & \vdash & P\{{}^{v'}/_{y}\} \\[2ex]
\textsc{Alloc} & \mathsf{assoc}\ \{\widetilde{(u, u')}\}, t, t' & \vdash & \mathsf{let}\ y = t(v)\ \mathsf{in}\ P \\
\longrightarrow & \mathsf{assoc}\ \{\widetilde{(u, u')}, (v, x'\sigma_{\mathrm{dv}})\}, t, t' \wedge D\sigma_{\mathrm{dv}} & \vdash & P\{{}^{x'\sigma_{\mathrm{dv}}}/_{y}\} \mid Q\sigma_{\mathrm{dv}}
\end{array}$$

with the side conditions for ALLOC that the process attached to t is $\mathsf{def}\ D\ \mathsf{in}\ Q \mid \mathsf{enter}\ x, x'$ (with x' defined in D), that there is no pair in $\{\widetilde{(u,u')}\}$ whose first element is v, that $dom(\sigma_{\mathrm{dv}}) = \mathsf{dv}[D]$, and that $range(\sigma_{\mathrm{dv}})$ consists of fresh and distinct names. The rule ASSOC represents the case where the value v in the query $t(v)$ already appears in a pair (v, v') present in the table; the associated value v' is substituted for the formal parameter y in P. The rule ALLOC represents the case where no value is associated with v; a new pair $(v, x'\sigma_{\mathrm{dv}})$ is immediately added to the table. We also adopt two symmetric rules ASSOC' and ALLOC'; we omit their formulations.

Chemical semantics provides a concise way to define concurrent reduction modulo structural equivalence. This presentation, however, is equivalent to a more traditional presentation based on reduction on processes (instead of chemical solutions). For processes, we define the relations of *structural equivalence* and of *reduction modulo structural equivalence* by a combination of heating, reduction, and cooling of chemical solutions:

$$
\begin{aligned}
P \equiv P' &\stackrel{\text{def}}{=} \emptyset \vdash \{P\} \rightleftharpoons^* \emptyset \vdash \{P'\} \\
P \rightarrow P' &\stackrel{\text{def}}{=} \emptyset \vdash \{P\} \rightleftharpoons^* \longrightarrow \rightleftharpoons^* \emptyset \vdash \{P'\}
\end{aligned}
$$

Relying on these definitions, everywhere else in this paper we use reduction on processes rather than on chemical solutions.

A.3 On several definitions of observational equivalence

Since every join-calculus process is also a sjoin-calculus process, we have two versions of observational equivalence for the join-calculus. One of them, which here we write $\approx_j$, is the standard relation of observational equivalence for the join-calculus, defined by considering all join-calculus contexts. The other one, which here we write $\approx_{sj}$, is the relation of observational equivalence for the sjoin-calculus, defined by considering all sjoin-calculus contexts. As the next lemma shows, these two relations on join-calculus processes coincide. Therefore, in the rest of this paper, we write $\approx$ instead of $\approx_j$ and $\approx_{sj}$; in proofs, we typically use the definition of $\approx$ as $\approx_{sj}$.

Lemma 1 *For all join-calculus processes P and Q, $P \approx_j Q$ if and only if $P \approx_{sj} Q$.*

Proof: Since every join-calculus context is also a sjoin-calculus context, it is evident that $\approx_{sj} \subseteq \approx_j$ on join-calculus processes. In order to establish that, conversely, $\approx_j \subseteq \approx_{sj}$, we can rely on a labeled bisimulation $\approx_l$ for the join-calculus [10]. The relation $\approx_l$ respects barbs, and its closure by application of evaluation contexts is easily established (see [10, 17]). Therefore, $\approx_l$ is at least as fine as $\approx_j$ and $\approx_{sj}$. Moreover, we have that $\approx_l = \approx_j$ [10]. (In contrast, in the sjoin-calculus, $\approx_l$ is strictly more discriminating than $\approx_{sj}$, as $\approx_{sj}$ equates the processes $\mathsf{def\ key}\ k\ \mathsf{in}\ x\langle\{v\}_k\rangle$ for all values v.) In summary, we have $\approx_j = \approx_l \subseteq \approx_{sj}$, so $\approx_j \subseteq \approx_{sj}$. $\square$

One could similarly argue that other possible variants in the definitions of observational-equivalence relations do not affect the resulting relations. In particular, one could allow ill-typed contexts in the definitions; intuitively, this corresponds to considering attackers that may not be well-typed. We believe that the relations of observational equivalence for the join-calculus and the sjoin-calculus are not affected by this change.

B Main proofs

This appendix contains the proofs for all the results stated in the paper; it is organized as follows. First, we provide some technical lemmas about equivalences and proof techniques in the join-calculus; these lemmas are not specific to security, but they form the basis for most proofs that follow. Then, we establish a few lemmas about the basic properties of components of our protocols. The core of this appendix consists of three independent series of results, about the correctness of the basic protocol, the correctness of the session-based protocol, and the soundness of the translations.

B.1 Technical lemmas

In order to prove our results in a modular manner, we rely on two co-inductive proof techniques, namely observational equivalence up to expansion and expansion up to expansion [41]. Lemmas 2 and 3 are technical lemmas that state sufficient conditions for relations to be included in the relations of observational equivalence and expansion, respectively; they are the basis for most of our proofs. The proofs of these lemmas are tedious but standard diagram-chasing arguments.

Lemma 2 (equivalence up to expansion) *To prove that a relation $\mathcal{R}$ is included in observational equivalence ($\mathcal{R} \subseteq \approx$), it suffices to prove that, for a fixed set of names $\mathcal{N}' \subseteq \mathcal{N}$ such that there are infinitely many names of every type in $\mathcal{N} \setminus \mathcal{N}'$, and for all processes P and Q such that $P\ \mathcal{R}\ Q$, we have the following properties:*

1. *If $P \downarrow_x$ then $Q \Downarrow_x$; conversely, if $Q \downarrow_x$ then $P \Downarrow_x$.*
2. *$C[P] \approx\mathcal{R}\approx C[Q]$ for every context $C[\cdot]$ of the form $\mathsf{def}\ D\ \mathsf{in}\ R \mid [\cdot]$ where $\mathsf{fv}[\mathsf{def}\ D\ \mathsf{in}\ R] \cap \mathcal{N}' = \emptyset$.*

3. *If $P \rightarrow P'$, then $Q \rightarrow^* Q'$ and $P' \succeq\mathcal{R}^{=}\approx Q'$ for some Q'.*

4. *If $Q \rightarrow Q'$, then $P \rightarrow^* P'$ and $P' \approx\mathcal{R}^{=}\preceq Q'$ for some P'.*

5. $(\mathsf{fv}[P] \cup \mathsf{fv}[Q]) \cap \mathcal{N}' = \emptyset$.

In the next lemma, and below, we write $\sim$ for the relation of strong equivalence; as usual, this relation is obtained from the definition of $\approx$ by substituting $\downarrow_x$ for $\Downarrow_x$ and $\rightarrow$ for $\rightarrow^*$. We omit the statement of the strong variant of Lemma 2 that is obtained by substituting $\downarrow_x$ for $\Downarrow_x$, $\rightarrow$ for $\rightarrow^*$, and $\sim$ for $\preceq$, $\succeq$, and $\approx$.

Lemma 3 (expansion up to expansion) *To prove that a relation $\mathcal{R}$ is included in expansion ($\mathcal{R} \subseteq \succeq$), it suffices to prove that, for a fixed set of names $\mathcal{N}' \subseteq \mathcal{N}$ such that there are infinitely many names of every type in $\mathcal{N} \setminus \mathcal{N}'$, and for all processes P and Q such that $P \mathrel{\mathcal{R}} Q$, we have the following properties:*

1. *If $P \downarrow_x$ then $Q \Downarrow_x$; conversely, if $Q \downarrow_x$ then $P \Downarrow_x$.*

2. *$C[P] \succeq\mathcal{R}\succeq C[Q]$ for every context $C[\,\cdot\,]$ of the form $\mathsf{def}\ D\ \mathsf{in}\ R \mid [\,\cdot\,]$ where $\mathsf{fv}[\mathsf{def}\ D\ \mathsf{in}\ R] \cap \mathcal{N}' = \emptyset$.*

3. *If $P \rightarrow P'$, then $Q \rightarrow^{=} Q'$ and $P' \succeq\mathcal{R}^{=}\succeq Q'$ for some Q'.*

4. *If $Q \rightarrow Q'$, then $P \rightarrow^* P'$ and $P' \succeq\mathcal{R}^{=}\sim Q'$ for some P'.*

5. $(\mathsf{fv}[P] \cup \mathsf{fv}[Q]) \cap \mathcal{N}' = \emptyset$.

In the special case where $P \mathrel{\mathcal{R}} Q$ implies $P\ (\rightarrow^*\succeq)^*\ Q$, the proof requirements can be further weakened: properties (1) and (4) are always true.

In order to establish the requirements of Lemmas 2 and 3, we can use relations stronger than those mentioned. For instance, since we have $\equiv\ \subseteq\ \sim\ \subseteq\ \succeq\ \subseteq\ \approx$, all proofs can be performed up to structural equivalence and up to strong equivalence. When we use Lemmas 2 and 3, we typically let $\mathcal{N}'$ collect names that are already bound in the processes being considered, for example *emit* and *recv*. We may omit the definition of $\mathcal{N}'$ and the check of property (5) when they are trivial.

These two lemmas lead to smaller relations and to simpler proofs. In particular, let us consider a relation $\mathcal{R}$ consisting of pairs of processes that are also related by some reduction that commutes with every other reduction; we call those reductions deterministic. For example, the relation could be:

$$\mathcal{R} \stackrel{\text{def}}{=} \{(C[\mathsf{if}\ u = u\ \mathsf{then}\ P\ \mathsf{else}\ Q],\ C[P]) \mid C[\,\cdot\,] \text{ an evaluation context}\}$$

We immediately obtain $\mathcal{R} \subseteq \succeq$ so, in proofs up to expansion, we can simplify processes by applying the deterministic reductions as soon as possible. For example, we may replace $\mathsf{if}\ u = u\ \mathsf{then}\ P\ \mathsf{else}\ Q$ with P in every evaluation context.

The next lemma is a direct consequence of our specification of association tables. It is useful for describing access to the tables in firewalls as expansion relations. It can be proved easily using Lemma 3.

We say that an association table $\mathsf{assoc}\ S, t, t'$ is bijective if whenever $(u, u') \in S$ and $(v, v') \in S$ we have $u = u'$ if and only if $v = v'$. (Every table that appears in a firewall is bijective.)

Lemma 4 *For all processes P and P', if $P \rightarrow P'$ by applying one of the chemical rules* Assoc, Assoc', Alloc, *or* Alloc' *with a bijective table in P, then we have $P \succeq P'$.*

The next lemma relates two formulations of equivalence relations. The first formulation is more compact, and appears in statements of our theorems. The second formulation is closed by application of evaluation contexts, and is used in the proofs and in the lemmas. The lemma is valid in both the join-calculus and the sjoin-calculus.

Lemma 5 *Let $\mathcal{R}$ be one of the relations $\sim$, $\succeq$, and $\approx$. Let X be a set of names, $\widetilde{X}$ be a set of tuples that partitions the names in X, $\{plug_{\widetilde{x}} \mid \widetilde{x} \in \widetilde{X}\}$ be a family of names indexed by $\widetilde{X}$, and $P_{\widetilde{X}}$ abbreviate the process $\prod_{\widetilde{x} \in \widetilde{X}} plug_{\widetilde{x}}\langle \widetilde{x} \rangle$.*

For all processes P_1, P_2 and definitions D_1, D_2 where none of the names $plug_{\widetilde{x}}$ occur free, and such that there are no free table names in $\mathsf{def}\ D_1\ \mathsf{in}\ P_1$ or $\mathsf{def}\ D_2\ \mathsf{in}\ P_2$, the two following statements are equivalent:

1. $\mathsf{def}\ D_1\ \mathsf{in}\ P_1 \mid P_{\widetilde{X}}\ \ \mathcal{R}\ \ \mathsf{def}\ D_2\ \mathsf{in}\ P_2 \mid P_{\widetilde{X}}$
2. *for all D and P such that $\mathsf{fv}[\mathsf{def}\ D\ \mathsf{in}\ P] \cap (\mathsf{dv}[D_1] \cup \mathsf{dv}[D_2]) \subseteq X$ and $\mathsf{dv}[D] \cap (\mathsf{dv}[D_1] \cup \mathsf{dv}[D_2]) = \emptyset$,*

$$\mathsf{def}\ D \wedge D_1\ \mathsf{in}\ P \mid P_1 \quad \mathcal{R} \quad \mathsf{def}\ D \wedge D_2\ \mathsf{in}\ P \mid P_2$$

Before proving Lemma 5, we illustrate its use by considering the compact sjoin-calculus equivalence that appears in Theorem 2. We apply Lemma 5 with $\widetilde{X} = \{\langle emit, recv \rangle\}$ and with $plug_{\langle emit, recv \rangle} = plug$. Up to structural equivalence, and for any processes P_1 and P_2 in the join-calculus, the following two statements are equivalent:

1. $\mathcal{E}nv\,[\,[\![P_1]\!]\,] \approx \mathcal{E}nv\,[\,[\![P_2]\!]\,]$

2. for every sjoin-calculus process $\mathsf{def}\ D\ \mathsf{in}\ P$ such that *emit* and *recv* are not defined in D,

$$\mathcal{E}[\,\mathsf{def}\ D\ \mathsf{in}\ P \mid [\![P_1]\!] \mid W\,] \;\approx\; \mathcal{E}[\,\mathsf{def}\ D\ \mathsf{in}\ P \mid [\![P_2]\!] \mid W\,]$$

This second statement deals with forms stable by reduction; D and P represent an arbitrary context, intuitively an attacker.

Next, we prove Lemma 5.

Proof: *(2) implies (1):* We obtain (1) as an instance of (2), up to strong equivalence, by letting $P = P_{\widetilde{X}}$, and letting D be empty (or more formally, any inert definition, such as $\mathsf{fresh}\ d$ where d does not appear elsewhere).

(1) implies (2): We proceed in two steps. Given D_1, D_2, P_1, and P_2 that satisfy the requirements of the lemma, and given D and P that satisfy the requirements of (2), we first construct processes Q_1 and Q_2 and prove that (1) implies $Q_1\ \mathcal{R}\ Q_2$. Then, we conclude by showing that $Q_1\ \mathcal{R}\ Q_2$ implies (2). Informally, Q_1 refines $\mathsf{def}\ D \wedge D_1\ \mathsf{in}\ P \mid P_1$ by interleaving relays for messages in P_1 sent on names defined in D, and by providing a name r to the context with the property that every message $r\langle\rangle$ enables a single message to be relayed.

Since $\mathcal{R}$ is closed under bijective renamings, we can assume that the names $plug_{\widetilde{x}}$ are not free in D or P. We start by partitioning D as $D \equiv D_c \wedge D_b \wedge D_t$ where D_c contains only channel definitions (join-rules $J \triangleright R$), D_b contains only $\mathsf{BitString}$ definitions ($\mathsf{keys}\ x^+, x^-$ or $\mathsf{fresh}\ d$), and D_t contains only table definitions (uids or assoc). Let $F \stackrel{\text{def}}{=} \mathsf{dv}[D_c]$. We say that a term M is an F-molecule when M is the parallel composition of messages $y\langle\widetilde{v}\rangle$ with $y \in F$. When M is an F-molecule, we write $\overline{M}$ for the term obtained by replacing each $y\langle\widetilde{v}\rangle$ with $\bar{y}\langle\widetilde{v}\rangle$. Similarly, we write $\overline{D}_c$ for the definition obtained by replacing each rule $J \triangleright R$ in D_c with $\overline{J} \triangleright R$.

Let r, t, κ, κ', ρ, z be distinct, fresh names; for every $y \in F$, let $\bar{y}$, $\hat{y}$, and $\widetilde{z}_y$ be additional distinct, fresh names. We set:

$$\begin{aligned}
D_F &\stackrel{\text{def}}{=} r\langle\rangle \triangleright t\langle\rangle \wedge \bigwedge_{y\in F} y\langle\widetilde{z}_y\rangle \mid \hat{y}\langle\bar{y}\rangle \mid r\langle\rangle \triangleright \bar{y}\langle\widetilde{z}_y\rangle \mid \hat{y}\langle\bar{y}\rangle \\
P_F &\stackrel{\text{def}}{=} \kappa\langle r\rangle \mid \prod_{y\in F} \hat{y}\langle\bar{y}\rangle \\
Q_i &\stackrel{\text{def}}{=} \mathsf{def}\ D_i \wedge \overline{D}_c \wedge D_b \wedge D_t \wedge D_F\ \mathsf{in}\ P_i \mid P \mid P_F
\end{aligned}$$

In order to establish $Q_1\ \mathcal{R}\ Q_2$, we use the following evaluation contexts:

$$\begin{aligned}
C[\,\cdot\,] &\stackrel{\text{def}}{=} \mathsf{def}\ D_b \wedge D_F\ \mathsf{in}\ [\,\cdot\,] \\
C_\rho[\,\cdot\,] &\stackrel{\text{def}}{=} \mathsf{def}\ \rho\langle\widetilde{x}\rangle \mid z\langle\rangle \triangleright t\langle\rangle \wedge \rho\langle\widetilde{x}\rangle \triangleright \mathsf{def}\ \overline{D}_c \wedge D_t\ \mathsf{in}\ P \mid P_F\ \mathsf{in}\ [\,\cdot\,] \\
C_X[\,\cdot\,] &\stackrel{\text{def}}{=} \mathsf{def}\ P_{\widetilde{X}} \triangleright \rho\langle\widetilde{x}\rangle\ \mathsf{in}\ [\,\cdot\,]
\end{aligned}$$

where $\widetilde{x}$ is a single tuple that contains all the names in X. Let $Z = \mathsf{0}$ or $Z = z\langle\rangle$ and consider the following reductions:

$$\begin{array}{ll} & C[C_\rho[C_X[Z \mid \mathsf{def}\ D_1\ \mathsf{in}\ P_1 \mid P_{\widetilde{X}}]]] \\ \rightarrow & C[C_\rho[C_X[Z] \mid \mathsf{def}\ D_1\ \mathsf{in}\ P_1 \mid \rho\langle\widetilde{x}\rangle]] \\ \rightarrow & C[C_\rho[C_X[Z]] \mid \mathsf{def}\ D_1 \wedge \overline{D}_c \wedge D_t\ \mathsf{in}\ P_1 \mid P \mid P_F] \sim Q_1 \end{array}$$

Note that the requirements of the lemma ensure that no α-conversion is needed. Since $\mathcal{R}$ is closed by application of any evaluation context, we have

$$C[C_\rho[C_X[Z \mid \mathsf{def}\ D_1\ \mathsf{in}\ P_1 \mid P_{\widetilde{X}}]]] \quad \mathcal{R} \quad C[C_\rho[C_X[Z \mid \mathsf{def}\ D_2\ \mathsf{in}\ P_2 \mid P_{\widetilde{X}}]]]$$

Next, we conclude the argument that $Q_1\ \mathcal{R}\ Q_2$, distinguishing two cases.

- If $\mathcal{R}$ is $\sim$ or $\succeq$, we take $Z = z\langle\rangle$. The reduction sequence above must be simulated by some reduction sequence

  $$C[C_\rho[C_X[Z \mid \mathsf{def}\ D_2\ \mathsf{in}\ P_2 \mid P_{\widetilde{X}}]]] \quad \rightarrow^n \quad Q_2'$$

  with $n \leq 2$ and $Q_1\ \mathcal{R}\ Q_2'$. We cannot have $Q_2' \Downarrow_t$, since we do not have $Q_1 \Downarrow_t$, so the reduction sequence must consume both $P_{\widetilde{X}}$ and $\rho\langle\widetilde{x}\rangle$, hence we must have $n = 2$ and

  $$Q_2' \equiv C[C_\rho[C_X[z\langle\rangle]]] \mid \mathsf{def}\ D_2 \wedge \overline{D}_c \wedge D_t\ \mathsf{in}\ P_2 \mid P \mid P_F] \sim Q_2$$

  so $Q_1\ \mathcal{R}\ Q_2$.

- If $\mathcal{R}$ is $\approx$, we take $Z = \mathsf{0}$. The two reductions detailed above are thus deterministic, so we have $C[C_\rho[C_X[\mathsf{def}\ D_1\ \mathsf{in}\ \mathsf{0} \mid P_1 \mid P_{\widetilde{X}}]]] \succeq Q_1$ and analogously for Q_2. We obtain $Q_1 \preceq\approx\succeq Q_2$, hence $Q_1 \approx Q_2$.

Next, we show that $Q_1\ \mathcal{R}\ Q_2$ implies (2). To this end, we let

$$\begin{array}{rcl} T_i & \stackrel{\text{def}}{=} & \mathsf{def}\ E_i \wedge \overline{D}_c \wedge D_F\ \mathsf{in}\ U_i \mid \overline{M}_i \mid P_F \\ R_i & \stackrel{\text{def}}{=} & \mathsf{def}\ E_i \wedge D_c\ \mathsf{in}\ U_i \mid M_i \end{array}$$

for any E_i, U_i, and M_i where M_i is an F-molecule, $\mathsf{dv}[E_i] \cap F = \emptyset$, and names in $B \stackrel{\text{def}}{=} \{r, t, \kappa, \kappa'\} \cup \{\bar{y}, \hat{y} \mid y \in F\}$ do not occur in E_i, U_i, M_i. We let $\mathcal{R}'$ be the relation that contains all pairs of processes (R_1, R_2) such that $T_1\ \mathcal{R}\ T_2$. We show that $\mathcal{R}'$ is included in $\mathcal{R}$. To complete the proof, we instantiate this result by setting E_i to $D_i \wedge D_b \wedge D_t$, U_i to $P_i \mid P$, and M_i to $\mathsf{0}$. Using structural equivalence, the hypothesis $T_1\ \mathcal{R}\ T_2$ becomes $Q_1\ \mathcal{R}\ Q_2$, and the conclusion $R_1\ \mathcal{R}\ R_2$ yields statement (2) of the lemma.

Reduction steps of T_i and R_i are closely related:

- If $T_2 \rightarrow T'_2$, then $T'_2 \equiv \mathsf{def}\ E'_2 \wedge \overline{D}_c \wedge D_F\ \mathsf{in}\ U'_2 \mid \overline{M'_2} \mid P_F$, where $\mathsf{dv}[E'_2] \cap F = \emptyset$, names in B do not occur in E'_2, U'_2, there exists M''_2 such that $M_2 \equiv M'_2 \mid M''_2$, and $R_2 \rightarrow \mathsf{def}\ E'_2\ \mathsf{in}\ U'_2 \mid M'_2 \stackrel{\text{def}}{=} R'_2$.

- If $R_1 \rightarrow R'_1$, then $R'_1 \equiv \mathsf{def}\ E'_1 \wedge D_c\ \mathsf{in}\ U'_1 \mid M'_1$, where $\mathsf{dv}[E'_1] \supseteq \mathsf{dv}[E_1]$, names in B do not occur in E'_1 or U'_1, $M_1 \equiv M'_1 \mid M''_1$, and M''_1 is composed of m messages that are all consumed by the reduction.

We use either Lemma 2 if $\mathcal{R}$ is $\approx$ or $\sim$, or Lemma 3 if $\mathcal{R}$ is $\succeq$. Taking $\mathcal{N}' = B \cup F$ makes properties (2) and (5) of those lemmas trivial. For property (1), we observe that the strong barbs of R_i and T_i are the same. If $\mathcal{R}$ is $\sim$ then R_1, T_1, T_2, and R_2 have the same strong barbs. Otherwise, if $R_1 \downarrow_x$ then $T_1 \downarrow_x$, $T_2 \Downarrow_x$, and $R_2 \Downarrow_x$ by using the correspondence of reductions in T_2 and in R_2.

For property (3), let $R_1 \rightarrow R'_1$, and let n be the total number of messages $y\langle\widetilde{v}\rangle$, with $y \in F$, that are consumed by the reduction to R'_1. We apply the context

$$C[\,\cdot\,] \stackrel{\text{def}}{=} \mathsf{def} \begin{array}{l} \quad \kappa'\langle r\rangle \triangleright t\langle\rangle \\ \wedge \quad \kappa'\langle r\rangle \triangleright \kappa\langle r\rangle \mid \prod_{i=1}^{n-m} r\langle\rangle \end{array} \mathsf{in}\ \mathsf{def}\ \kappa\langle r\rangle \triangleright \kappa'\langle r\rangle\ \mathsf{in}\ [\,\cdot\,]$$

to the equivalence $T_1 \mathrel{\mathcal{R}} T_2$. By construction, $C[T_1] \rightarrow^{n-m+3} C[\mathsf{0}] \mid T'_1 \sim T'_1$ where $T'_1 \equiv \mathsf{def}\ \overline{E'_1} \wedge D_F\ \mathsf{in}\ U'_1 \mid \overline{M'_1} \mid P_F$. The closure property of $\mathcal{R}$ gives us $C[T_1] \mathrel{\mathcal{R}} C[T_2]$, so there must be a matching reduction $C[T_2] \rightarrow^{n'} T''_2$, with $T'_1 \mathrel{\mathcal{R}} T''_2$.

Furthermore we must not have $T''_2 \Downarrow_t$, as we do not have $T'_1 \Downarrow_t$, so among the n' reductions there must be two reductions that use the definitions of $C[\,\cdot\,]$ to consume the messages $\kappa\langle r\rangle$ in T_2 and the resulting $\kappa'\langle r\rangle$. These produce $n-m$ copies of $r\langle\rangle$, so there must also be exactly $n-m$ reductions that use the definition D_F to consume every message $r\langle\rangle$. (Otherwise the $\kappa'\langle r\rangle \triangleright t\langle\rangle$ or $r\langle\rangle \triangleright t\langle\rangle$ rules could produce a $t\langle\rangle$.) The remaining $n'' \stackrel{\text{def}}{=} n' + m - n - 2$ reductions must be reductions that involve only $E_2 \wedge \overline{D}_c$ or U_2 , which must correspond one-to-one to reductions of R_2, as above; the D_F reductions correspond simply to structural rearrangements, moving messages from U_2 to M_2.

Therefore, there must be some E'_2, U'_2, and M'_2 such that $\mathsf{dv}[E'_2] \cap F = \emptyset$, that names in B do not occur in E'_2, U'_2, M'_2, that M'_2 is an F-molecule, and that $R_2 \rightarrow^{n''} \mathsf{def}\ E'_2\ \mathsf{in}\ U'_2 \mid M'_2 \stackrel{\text{def}}{=} R'_2$, and $T''_2 \equiv C[\mathsf{0}] \mid T'_2$, where $T'_2 \equiv \mathsf{def}\ E'_2 \wedge \overline{D}_c \wedge D_F\ \mathsf{in}\ U'_2 \mid \overline{M'_2} \mid P_F$. We therefore have $T'_1 \mathrel{\mathcal{R}} T''_2 \sim T'_2$, hence $R'_1 \mathrel{\mathcal{R}'} R'_2$.

This completes the proof of property (3) if $\mathcal{R}$ is $\approx$; the symmetrical argument yields property (4) when $\mathcal{R}$ is $\approx$ or $\succeq$. In the other cases, we must have $n' = n - m + 3$ if $\mathcal{R}$ is $\sim$, and $n' \leq n - m + 3$ if $\mathcal{R}$ is $\succeq$; hence $n'' = 1$, and $n'' \leq 1$, respectively. $\square$

B.2 Basic properties of the public network

We are interested in equivalences in contexts that represent a public network with the noise-generating process W. By definition, we have:

$$\mathcal{E}[\,\cdot \mid W\,] \quad = \quad \begin{array}{l}\mathsf{def}\ recv\langle\kappa\rangle \mid emit\langle m\rangle \triangleright \kappa\langle m\rangle\ \mathsf{in} \\ [\cdot] \mid \mathsf{repl}\ (\mathsf{def\ fresh}\ m\ \mathsf{in\ repl}\ emit\langle m\rangle)\end{array}$$

Rearranging a parallel composition, we also have:

$$\mathcal{E}nv\,[\,\cdot\,] \;\equiv\; \mathcal{E}\,[\,plug\langle emit, recv\rangle \mid [\cdot] \mid W\,]$$

We define generalizations $\mathcal{E}_{\dots}[\,\cdot\,]$ and $W_{\dots}$ of $\mathcal{E}[\,\cdot\,]$ and W, respectively:

$$\begin{array}{rcl}\mathcal{E}_{\substack{x_1^{\pm},\dots,x_m^{\pm},\\ k_1,\dots,k_n,\\ d_1,\dots,d_p}}[\,\cdot\,] & \stackrel{\text{def}}{=} & \begin{array}{ll}\mathsf{def} & \bigwedge_{i\in 1..m}\mathsf{keys}\ x_i^+, x_i^- \\ \wedge & \bigwedge_{j\in 1..n}\mathsf{key}\ k_j \\ \wedge & \bigwedge_{l\in 1..p}\mathsf{fresh}\ d_l \qquad \mathsf{in} \quad \mathcal{E}[\,\cdot\,]\end{array} \\ W_{d_1,\dots,d_p} & \stackrel{\text{def}}{=} & W \mid \prod_{l\in 1..p}\mathsf{repl}\ emit\langle d_l\rangle\end{array}$$

We abbreviate $\mathcal{E}_{x_1^{\pm},\dots,x_m^{\pm}}[\,\cdot\,]$ to $\mathcal{E}_{X^{\pm}}[\,\cdot\,]$ when $X^{\pm} = \{x_1^{\pm},\dots,x_m^{\pm}\}$. We always assume in the following that names that are bound in contexts $\mathcal{E}_{\dots}[\,]$ and $\mathcal{E}nv\,[\,]$—that is, $recv$, $emit$, and every index variable in $\mathcal{E}_{\dots}[\,]$—are not masked by other definitions that appear in the context.

The use of a substitution of a ciphertext for a name enables us to capture the notion that a certain context has access to the ciphertext but not to its components. This substitution is important for achieving stability by reduction, because the ciphertext may be intercepted in the course of reduction. The next lemma says that the ciphertext cannot be distinguished from noise without knowledge of the decryption key.

Lemma 6 *For all Q, k, d, and $\widetilde{v}$ such that k does not occur in Q and d does not occur in $\widetilde{v}$, we have:*

$$\mathcal{E}_k\left[\,Q\left\{{}^{\{\widetilde{v}\}_k}/_d\right\}\right] \;\sim\; \mathcal{E}_d\,[\,Q\,]$$

Proof: Let $\mathcal{R}$ be the relation that contains all pairs of processes (P_1, P_2) related in the lemma. We prove that $\mathcal{R} \subseteq \sim$ by using the strong variant of Lemma 2 with the set $\mathcal{N}' = \{emit, recv, k, d\}$. The sufficient properties are mostly straightforward except when substitutions and reductions interact:

1. The property of strong barbs is immediate to check, because the substitution does not operate on channel names.

2. The relation $\mathcal{R}$ is closed by application of an evaluation context $C[\cdot]$ followed by structural rearrangement, provided that the free names of this context do not clash with $\mathcal{N}'$ ($\mathsf{fv}[C[0]] \cap \mathcal{N}' = \emptyset$). This restriction guarantees that the free names of $C[\cdot]$ are not captured by the contexts $\mathcal{E}_k\left[[\cdot]\{^{\{\widetilde{v}\}_k}/_d\}\right]$ and $\mathcal{E}_d[\cdot]$, so $C[\cdot]$ can be commuted with these contexts.

3. In order to establish the strong bisimulation property, we show that the substitution of $\{\widetilde{v}\}_k$ for d and the change from $\mathcal{E}_k[\,]$ to $\mathcal{E}_d[\,]$ commute with every reduction. We partition the reductions $P_1 \rightarrow P_1'$ and $P_2 \rightarrow P_2'$ according to the chemical reduction rule that is used in the reduction.

   Red: Both d and $\{\widetilde{v}\}_k$ are values of type $\mathsf{BitString}$, while these reductions depend only on channel names.

   Compare, Uids, Alloc, Assoc, Alloc', Assoc': Since k does not occur in Q, $\{\widetilde{v}\}_k$ does not appear in Q, so the substitution is a bijection on the values that appear in Q. Therefore, the outcome of every reduction that compares values is left unchanged by the substitution.

   Decrypt: Since k does not occur in Q, each decryption attempt in Q behaves like the corresponding decryption attempt in $Q\left\{^{\{\widetilde{v}\}_k}/_d\right\}$.
   $\square$

The next lemma describes the behavior of the filtering reception $\mathsf{let}\ \widetilde{y} = \mathsf{filter}\ x^-\ \mathsf{in}\ Q$ in a network that provides a message m.

Lemma 7 *Let $C[\cdot]$ be the context $\mathcal{E}_{x^\pm}[\mathsf{def}\ D\ \mathsf{in}\ P \mid \cdot\,]$, and let R be the process $\mathsf{decrypt}\ m\ \mathsf{using}\ x^-\ \mathsf{to}\ \widetilde{y}\ \mathsf{in}\ Q\ \mathsf{else}\ (emit\langle m\rangle \mid recv\langle\kappa\rangle)$. For all D, P, Q, m, we have the sequence of reductions:*

$$\begin{array}{rl} & C[emit\langle m\rangle \mid \mathsf{let}\ \widetilde{y} = \mathsf{filter}\ x^-\ \mathsf{in}\ Q] \\ \equiv & C[\mathsf{def}\ \kappa\langle m\rangle \triangleright R\ \mathsf{in}\ emit\langle m\rangle \mid recv\langle\kappa\rangle] \\ \rightarrow & C[\mathsf{def}\ \kappa\langle m\rangle \triangleright R\ \mathsf{in}\ \kappa\langle m\rangle] \\ \rightarrow & C[\mathsf{def}\ \kappa\langle m\rangle \triangleright R\ \mathsf{in}\ R] \end{array}$$

In case m is of the form $\{\widetilde{v}\}_{x^+}$ for some $\widetilde{v}$ of the same length as $\widetilde{y}$, decryption in R succeeds, and the sequence continues as follows:

$$C[\mathsf{def}\ \kappa\langle m\rangle \triangleright R\ \mathsf{in}\ R] \ \ \rightarrow\equiv\ \ C[Q\{^{\widetilde{v}}/_{\widetilde{y}}\} \mid \mathsf{def}\ \kappa\langle m\rangle \triangleright R\ \mathsf{in}\ 0] \ \sim\ C[Q\{^{\widetilde{v}}/_{\widetilde{y}}\}]$$

In all other cases, decryption in R fails, and yields back the initial process:

$$\begin{array}{rcl} C[\mathsf{def}\ \kappa\langle m\rangle \triangleright R\ \mathsf{in}\ R] & \rightarrow & C[\mathsf{def}\ \kappa\langle m\rangle \triangleright R\ \mathsf{in}\ emit\langle m\rangle \mid recv\langle\kappa\rangle] \\ & \equiv & C[emit\langle m\rangle \mid \mathsf{let}\ \widetilde{y} = \mathsf{filter}\ x^-\ \mathsf{in}\ Q] \end{array}$$

In any case, the last two steps of reduction ($\rightarrow\rightarrow$) are deterministic, and can thus be replaced with expansion ($\succeq$).

Proof: All the reductions above are obtained from the operational semantics defined in Appendix A. We merely comment every stage.

Before the first reduction, the derived construct filter is replaced with its actual definition, and the emission $emit\langle m\rangle$ is placed close to the reception offer $recv\langle\kappa\rangle$ using structural equivalence. The first reduction uses the definition of the network embedded in $C[\,\cdot\,]$; it consumes the emission $emit\langle m\rangle$ and the reception offer issued by the filter, and releases the process $\kappa\langle m\rangle$. This reduction selects a particular value m, but there may be other choices, as P may offer other messages $emit\langle m'\rangle$. The second reduction consumes the message $\kappa\langle m\rangle$ and triggers the guarded process R; the third reduction is the decryption attempt. For a given value of m, these last two reductions are deterministic.

In case decryption succeeds, the remains of the filtering process consists of the definition of κ, but as κ does not appear anywhere except in the guarded process R, this definition cannot be triggered any more, and is discarded up to strong equivalence. In case decryption fails, the processes $emit\langle m\rangle$ and $recv\langle\kappa\rangle$ are released. These two processes are identical to those that were consumed by the first reduction, and thus lead back to the initial stage of the reduction sequence. $\square$

In the whole section, when the process Q and the names x^- and $\widetilde{y}$ are clear from context, we use the abbreviations $K_{\langle m\rangle}$ and K_m for the derivatives of $emit\langle m\rangle \mid \mathsf{let}\ \widetilde{y} = \mathsf{filter}\ x^-\ \mathsf{in}\ Q$. Thus, we take:

$$\begin{aligned} K_{\langle m\rangle} &\stackrel{\text{def}}{=} \mathsf{def}\ \kappa\langle m\rangle \triangleright R\ \mathsf{in}\ \kappa\langle m\rangle \\ K_m &\stackrel{\text{def}}{=} \mathsf{def}\ \kappa\langle m\rangle \triangleright R\ \mathsf{in}\ R \end{aligned}$$

where R is $\mathsf{decrypt}\ m\ \mathsf{using}\ x^-\ \mathsf{to}\ \widetilde{y}\ \mathsf{in}\ Q\ \mathsf{else}\ (emit\langle m\rangle \mid recv\langle\kappa\rangle)$. With this convention, we can rewrite the first two reductions of the lemma above as:

$$C[emit\langle m\rangle \mid \mathsf{let}\ \widetilde{y} = \mathsf{filter}\ x^-\ \mathsf{in}\ Q] \rightarrow C[K_{\langle m\rangle}] \rightarrow C[K_m]$$

Lemma 8 says that, when both emission and filtered reception are replicated, communication occurs.

Lemma 8 *Let $C[\,\cdot\,]$ be the following sjoin-calculus context:*

$$C[\,\cdot\,] \stackrel{\text{def}}{=} \mathcal{E}_{x^\pm}\left[\begin{array}{lll} & & P \mid [\,\cdot\,] \mid W \\ \mathsf{def}\ D\ \mathsf{in} & \mid & (\mathsf{repl}\ emit\langle\{\widetilde{v}\}_{x^+}\rangle) \\ & \mid & (\mathsf{repl}\ \mathsf{let}\ \widetilde{y} = \mathsf{filter}\ x^-\ \mathsf{in}\ Q) \end{array}\right]$$

For every D, P, Q, and $\widetilde{v}$ such that the tuples $\widetilde{v}$ and $\widetilde{y}$ have the same length, we have the expansion relation $C[0] \succeq C[Q\{\widetilde{v}/\widetilde{y}\}]$.

Proof: We compare reductions on $C[\mathsf{0}]$ and on $C[Q\{^{\widetilde{v}}/_{\widetilde{y}}\}]$. We have

$$C[\mathsf{0}] \quad \equiv \quad C[\mathit{emit}\langle\{\widetilde{v}\}_{x^+}\rangle \mid \mathsf{let}\ \widetilde{y} = \mathsf{filter}\ x^-\ \mathsf{in}\ Q]$$

The two processes above are structurally equivalent because they differ only by a copy of a replicated emission and a copy of a replicated reception out of the context $C[\,\cdot\,]$. After unfolding these two processes, the successful sequence of Lemma 7 is enabled:

$$C[\mathsf{0}] \quad \equiv\rightarrow\succeq \quad C[Q\{^{\widetilde{v}}/_{\widetilde{y}}\}]$$

Let $\mathcal{R}$ be the relation that contains the pairs of processes $(C[\mathsf{0}], C[Q\{^{\widetilde{v}}/_{\widetilde{y}}\}])$ for every D, P, Q, and $\widetilde{v}$. We prove that $\mathcal{R} \subseteq \succeq$ by Lemma 3, in the special case where $P_1\ \mathcal{R}\ P_2$ implies $P_1 \rightarrow^*\succeq P_2$, and for $\mathcal{N}' = \{x^+, x^-, \mathit{emit}, \mathit{recv}\}$.

To establish property (2), let us consider the application of a context def D' in $P' \mid [\,\cdot\,]$ to processes related by $\mathcal{R}$. Assuming that names in $\mathcal{N}'$ do not occur in D' and P', we have

$$\mathsf{def}\ D'\ \mathsf{in}\ P' \mid C[Q\{^{\widetilde{v}}/_{\widetilde{y}}\}] \quad \equiv \quad C'[Q\{^{\widetilde{v}}/_{\widetilde{y}}\}]$$

where $C'[\,\cdot\,]$ is obtained from $C[\,\cdot\,]$ by substituting $P' \mid P$ for P and $D' \wedge D$ for D (with α-conversion on $\mathsf{dv}[D']$ as needed). Similarly, we have $\mathsf{def}\ D'\ \mathsf{in}$ $P' \mid C[\mathsf{0}] \equiv C'[\mathsf{0}]$ and therefore

$$\mathsf{def}\ D'\ \mathsf{in}\ P' \mid C[\mathsf{0}] \quad \equiv\mathcal{R}\equiv \quad \mathsf{def}\ D'\ \mathsf{in}\ P' \mid C[Q\{^{\widetilde{v}}/_{\widetilde{y}}\}]$$

To establish property (3), let us assume that $C[\mathsf{0}] \rightarrow P'$. This reduction step is due to reduction within $C[\,\cdot\,]$, so we have that $P' \equiv C'[\mathsf{0}]$ for some evaluation context $C'[\,\cdot\,]$, and the same reduction within $C[\,\cdot\,]$ yields $C[Q\{^{\widetilde{v}}/_{\widetilde{y}}\}] \rightarrow C'[Q\{^{\widetilde{v}}/_{\widetilde{y}}\}]$. Moreover, the replicated processes within $C[\,\cdot\,]$ cannot be consumed by this reduction, thus the resulting processes can be rearranged into processes in $\mathcal{R}$ using structural equivalence: there is a context $C''[\,\cdot\,]$ such that

$$C'[\mathsf{0}] \;\equiv\; C''[\mathsf{0}] \;\;\mathcal{R}\;\; C''[Q\{^{\widetilde{v}}/_{\widetilde{y}}\}] \;\equiv\; C'[Q\{^{\widetilde{v}}/_{\widetilde{y}}\}]$$

□

The following lemma says that, when a replicated emission and a filtered reception are the only processes with access to a shared key k, communication occurs.

Lemma 9 *For all D, P, Q, $\widetilde{v}$, and $\widetilde{y}$ in which k does not occur free and such that $\widetilde{v}$ and $\widetilde{y}$ have the same lengths and d does not occur free in $\widetilde{v}$, we have:*

$$\begin{aligned} & \mathcal{E}_k\left[\,(\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid \mathsf{let}\ \widetilde{y} = \mathsf{filter}\ k\ \mathsf{in}\ Q)\left\{^{\{\widetilde{v}\}_k}/_d\right\}\right] \\ \succeq\;\; & \mathcal{E}_d\left[\,\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid Q\{^{\widetilde{v}}/_{\widetilde{y}}\}\,\right] \end{aligned}$$

Proof: Let $\mathcal{R}$ be the relation that contains all the pairs of processes (P_1, P_2) described in the statement of the lemma. We use Lemma 3 with the set $\mathcal{N}' = \{d, k, emit, recv\}$ to show that $\mathcal{R} \subseteq \succeq$. As in the proof of Lemma 6, property (2) of Lemma 3 holds. In order to establish property (3), we compare the possible reductions of two processes P_1 and P_2 in the relation $\mathcal{R}$.

- The reductions that consume only processes present in P or W_d are treated as in the proof of Lemma 6.

- There is only one kind of reduction enabled in P_1 but not in P_2. This reduction consists of a join of the reception offer $recv\langle\kappa\rangle$ issued by the filtering process $\mathsf{let}\ \widetilde{y} = \mathsf{filter}\ k\ \mathsf{in}\ Q$ with a message $emit\langle m\{^{\{\widetilde{v}\}_k}/_d\}\rangle$ issued by either P or W_d. Using structural equivalence, we can write this reduction as

$$\begin{aligned} P_1 &\equiv \mathcal{E}_k\left[(\mathsf{def}\ D'\ \mathsf{in}\ P' \mid W_d \mid emit\langle m\rangle \mid \mathsf{let}\ \widetilde{y} = \mathsf{filter}\ k\ \mathsf{in}\ Q) \left\{^{\{\widetilde{v}\}_k}/_d\right\} \right] \\ &\rightarrow \mathcal{E}_k\left[\left(\mathsf{def}\ D'\ \mathsf{in}\ P' \mid W_d \mid K_{\langle m\rangle}\right) \left\{^{\{\widetilde{v}\}_k}/_d\right\} \right] \;=\; P_1' \end{aligned}$$

  for some D', P', m in which k does not occur. We proceed by cases on the form of m, as described in Lemma 7.

  - If $m = d$, then the decryption attempt will succeed. Let $P_1' \succeq P_1''$ be the expansion relation of Lemma 7 for that case. We can unfold a new copy of the message $emit\langle d\rangle$ out of the process W_d in P_1'' to replace the consumed message, and thus obtain

$$P_1' \;\succeq\; P_1'' \;\equiv\; \mathcal{E}_k\left[\left(\mathsf{def}\ D'\ \mathsf{in}\ P' \mid W_d \mid Q\{^{\widetilde{v}}/_{\widetilde{y}}\}\right) \left\{^{\{\widetilde{v}\}_k}/_d\right\} \right]$$

    The name k does not occur under the substitution any more; by Lemma 6 we derive $P_1'' \sim P_2$ and thus $P_1' \succeq P_2$.

  - If $m \neq d$, then the hypothesis on k implies that $m\{^{\{\widetilde{v}\}_k}/_d\}$ cannot be $\{\widetilde{w}\}_k$ for any $\widetilde{w}$. Therefore, the decryption attempt will fail and yield back the initial process: by Lemma 7 we have $P_1' \succeq\equiv P_1$ and thus $P_1' \succeq\mathcal{R}\ P_2$.

  In both cases, property (3) of Lemma 3 holds.

In P_1, the subprocess W_d is structurally equivalent to $W_d \mid emit\langle d\rangle$; when we replace d with $\{\widetilde{v}\}_k$, the message $emit\langle\{\widetilde{v}\}_k\rangle$ enables the successful decryption sequence $P_1 \rightarrow^* \succeq P_2$ described above. Therefore, we are in the special case of Lemma 3 where $\mathcal{R} \subseteq (\rightarrow^*\succeq)^*$, so properties (1) and (4) are immediately true. $\square$

B.3 Correctness of the basic protocol

We first extend our notations in order to represent the intermediate states of the receiving process of the basic protocol, R_x. We define:

$$R_x\{\widetilde{u}\}[G] \quad \stackrel{\text{def}}{=} \quad \mathsf{def\ uids}\ t_x\{\widetilde{u}\}\ \mathsf{in}\ G \mid \mathsf{repl}\ F_x$$

Intuitively, the process G represents intermediate states of the receiving process in the course of processing arriving messages. In particular, G is $\mathsf{0}$ when the receiving process is in its initial state: since R_x is $\mathsf{def\ uids}\ t_x\ \mathsf{in\ repl}\ F_x$, we have that $R_x \equiv R_x\{\}[\mathsf{0}]$.

In the proofs about the basic protocol, whenever we write $R_x\{\widetilde{u}\}[G]$, we assume that G is the parallel composition of derivatives of processes of the form $emit\langle m\rangle \mid F_x$. We apply Lemma 7 in order to describe those derivatives. In applying Lemma 7, we let

$$Q_{c,\widetilde{v}} \quad \stackrel{\text{def}}{=} \quad \mathsf{if\ not\ tset}\ t_x(c)\ \mathsf{then}\ x^\circ\langle\widetilde{v}\rangle$$

and take $Q = Q_{c,\widetilde{y}}$, so G is the parallel composition of processes of the forms $K_{\langle m\rangle}$, K_m, and $Q_{c,\widetilde{v}}$. (The process $Q_{c,\widetilde{v}}$ is obtained from a process K_m by a successful decryption in the case $m = \{c,\widetilde{v}\}_{x^+}$.) For example, G might be $K_{\langle m\rangle} \mid K_{m'}$, with m and m' arbitrary values of type $\mathsf{BitString}$.

Because the basic protocol relies on replicated emissions and receptions, G may contain more than one component of the form $Q_{c,\widetilde{v}}$. Lemma 10 states that such components can be discarded (up to strong equivalence) whenever the value c is already registered in the set t_x of unique identifiers. Its proof is immediate.

Lemma 10 *For all D, P, $\widetilde{u}$, $\widetilde{v}$, c we have the strong equivalence:*

$$\mathsf{def\ uids}\ t_x\{\widetilde{u},c\} \wedge D\ \mathsf{in}\ (P \mid Q_{c,\widetilde{v}}) \quad \sim \quad \mathsf{def\ uids}\ t_x\{\widetilde{u},c\} \wedge D\ \mathsf{in}\ P$$

Lemma 11 states that what is left after a successful run of the protocol is indistinguishable from noise, and can be discarded (up to strong equivalence).

Lemma 11 *For all D, P, G, $\widetilde{u}$, $\widetilde{v}$ such that the length of $\widetilde{v}$ is the arity of x, the name x^- occurs only as a decryption key within G and F_x, the name c does not occur in D, P, G, $\widetilde{u}$, $\widetilde{v}$, and the name d does not occur in $\widetilde{v}$, we have the strong equivalence:*

$$\begin{aligned} &\mathcal{E}_{x^\pm,c}\left[\,(\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid R_x\{\widetilde{u},c\}[G])\left\{{}^{\{c,\widetilde{v}\}_{x^+}}/_{d}\right\}\right] \\ \sim\ \ &\mathcal{E}_{x^\pm,d}\left[\,\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid R_x\{\widetilde{u}\}[G]\,\right] \end{aligned}$$

Proof: As in the proof of Lemma 9, the use of a substitution of a ciphertext for a name arises because a certain context has access to the ciphertext but not to its components. Our proof here uses heavier notations because we are not in the special case of Lemma 3.

Let $\mathcal{R}$ be the relation that contains all pairs of processes (P_1, P_2) for which the lemma claims $P_1 \sim P_2$. We prove that $\mathcal{R} \subseteq \sim$ by applying the strong variant of Lemma 2 with $\mathcal{N}' = \{x^+, x^-, c, d, \mathit{emit}, \mathit{recv}\}$.

As in the proof of Lemma 6, we prove the strong bisimulation property by a case analysis on the chemical reduction rule being used in the reduction step. Except for the comparisons and the decryption attempts that involve $\{c, \widetilde{v}\}_{x^+}$ in P_1 and d in P_2, all reductions are the same on both sides of the relation $\mathcal{R}$, and lead to processes related by $\mathcal{R}$ after structural rearrangement. We analyze comparison and decryption attempts much as in Lemma 6.

Compare, Uids, Alloc, Assoc, Alloc', Assoc': The hypothesis on the occurrences of c implies that $\{c, \widetilde{v}\}_{x^+}$ cannot appear before the substitution; therefore, that hypothesis guarantees that the substitution does not affect the result of comparisons.

Decrypt: In the case where any value other than x^- is used as decryption key, rule Decrypt is handled like comparison attempts. We describe the case where x^- is used as decryption key in more detail. The hypotheses ensure that decryptions with key x^- can happen only in the subprocess G of P_1, in case $G \equiv G' \mid K_d$ for some G' (before substitution). In this case, the decryption of $\{c, \widetilde{v}\}_{x^+}$ in P_1 succeeds, while the decryption of d in P_2 always fails.

The successful decryption reduction $P_1 \to P_1'$ replaces $K_{\{c,\widetilde{v}\}_{x^+}}$ with $Q_{c,\widetilde{v}}$, and the latter process can be discarded up to strong equivalence, by Lemma 10:

$$\begin{array}{rcl} P_1 & \equiv & \mathcal{E}_{x^\pm,c}\left[\left(\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid R_x\{\widetilde{u},c\}[G' \mid K_d]\right)\left\{{}^{\{c,\widetilde{v}\}_{x^+}}/_{d}\right\}\right] \\ & \to\sim & \mathcal{E}_{x^\pm,c}\left[\left(\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid R_x\{\widetilde{u},c\}[G']\right)\left\{{}^{\{c,\widetilde{v}\}_{x^+}}/_{d}\right\}\right] \end{array}$$

On the other hand, a failing decryption reduction $P_2 \to P_2'$ releases the subprocesses F_x and $\mathit{emit}\langle d\rangle$, which can be folded into R_x and W_d respectively:

$$\begin{array}{rcl} P_2 & \equiv & \mathcal{E}_{x^\pm,d}\left[\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid R_x\{\widetilde{u}\}[G' \mid K_d]\right] \\ & \to & \mathcal{E}_{x^\pm,d}\left[\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid R_x\{\widetilde{u}\}[G' \mid F_x \mid \mathit{emit}\langle d\rangle]\right] \\ & \equiv & \mathcal{E}_{x^\pm,d}\left[\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid R_x\{\widetilde{u}\}[G']\right] \end{array}$$

We obtain the relation $P_1' \sim\mathcal{R}\equiv P_2'$, and hence the strong bisimulation property. The other properties are immediate. □

Lemma 12 concerns the progress of pending communication.

Lemma 12 *Let $C[\,\cdot\,]$ be a context of the form*

$$C[\,\cdot\,] \quad \stackrel{def}{=} \quad \mathcal{E}_{x^\pm,c}\left[(\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid [\,\cdot\,]) \left\{ {}^{\{c,\widetilde{v}\}_{x^+}}/_{d} \right\} \right]$$

where the substitution $\left\{ {}^{\{c,\widetilde{v}\}_{x^+}}/_{d} \right\}$ *operates both on the context* $\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid [\,\cdot\,]$ *and on the process put in the hole of this context. Assume that the length of the tuple $\widetilde{v}$ is the arity of x, that d is not bound in D, that c and x^- do not occur in D, P, $\widetilde{u}$, and $\widetilde{v}$, and that d does not occur in $\widetilde{v}$. Let G be a parallel composition of derivatives of $emit\langle m\rangle \mid F_x$ for values m where c and x^- occur only in processes $Q_{c,\widetilde{v}}$ (but d may occur in m).*

We have the expansion relation:

$$C[R_x\{\widetilde{u}\}[G]] \quad \succeq \quad C[R_x\{\widetilde{u}, c\}[G] \mid x^\circ\langle\widetilde{v}\rangle]$$

Proof: Let $\mathcal{R}$ be the relation that contains all pairs $(P_1[G], P_2[G])$ for which the lemma claims $P_1[G] \succeq P_2[G]$.

The process $P_1[G]$ contains as subprocesses the replicated emission repl $emit\langle\{c,\widetilde{v}\}_{x^+}\rangle$ (within $W_d \left\{ {}^{\{c,\widetilde{v}\}_{x^+}}/_{d} \right\}$) and the replicated filtering reception $\mathsf{repl\ let}\ \widetilde{y} = \mathsf{filter}\ x^-\ \mathsf{in}\ Q$ (within $R_x\{\widetilde{u}\}$). Lemma 8 implies the expansion relation:

$$P_1[G] \quad \succeq \quad P_1[G \mid Q_{c,\widetilde{v}}]$$

Provided that there is at least one copy of the process $Q_{c,\widetilde{v}}$, a single reduction registers the value c in the table in $R_x\{\widetilde{u}\}[\,\cdot\,]$, and thus delivers the message $x^\circ\langle\widetilde{v}\rangle$:

$$P_1[G \mid Q_{c,\widetilde{v}}] \;=\; C[R_x\{\widetilde{u}\}[G \mid Q_{c,\widetilde{v}}]] \;\rightarrow\; C[R_x\{\widetilde{u}, c\}[G \mid x^\circ\langle\widetilde{v}\rangle]] \;\equiv\; P_2[G]$$

Combining these relations, we obtain $P_1[G] \succeq\rightarrow\equiv P_2[G]$. Hence, we prove $\mathcal{R} \subseteq \succeq$ by applying the special case of Lemma 3 where $\mathcal{R} \subseteq (\rightarrow^*\succeq)^*$, with $\mathcal{N}' = \{x^+, x^-, c, d\}$. Properties (2) and (5) hold by construction of $\mathcal{R}$. In order to establish property (3), we partition the reductions $P_1[G] \rightarrow P_1'$. Except for the last subcase below, every such reduction is directly matched by a reduction $P_2[G] \rightarrow P_2'$ with $P_1' \equiv\mathcal{R}\equiv P_2'$. (In each case, establishing that $P_1' \equiv\mathcal{R}\equiv P_2'$ requires the construction of a new context $C[\,\cdot\,]$ and process G.)

1. The reduction uses a join-pattern in D, is internal to P, or uses the join-pattern of the context $\mathcal{E}_{x^\pm,c}[\,\cdot\,]$ to consume a message $emit\langle m\rangle$ in $P \mid W_d$ and a message $recv\langle\kappa\rangle$ in P. Only D and P are affected.

2. The reduction uses the join-pattern of the context $\mathcal{E}_{x^\pm,c}[\,\cdot\,]$ to consume a message $emit\langle m\rangle$ in $P \mid W_d$ and a message $recv\langle\kappa\rangle$ in $R_x\{\widetilde{u}\}[G]$. This reception request is defined in a process F_x. As a result, G is replaced with $G \mid K_{\langle m\rangle}$ for a $\mathsf{BitString}$ value m that does not contain c or x^- (before substitution).

3. The reduction is internal to G. By definition of G, several subcases may occur. In analyzing them, we write P_i for P_1 and P_2.

   (a) A continuation is received: $P_i[G' \mid K_{\langle m\rangle}] \to P_i[G' \mid K_m]$.

   (b) A decryption fails: $P_i[G' \mid K_m] \to P_i[G' \mid F_x \mid emit\langle m\rangle]$ for $i = 1, 2$. Using structural equivalence, F_x can be folded in $R_x\{\widetilde{u}\}[\cdot]$ and $emit\langle m\rangle$ can be added to P.

   (c) A decryption succeeds: $P_i[G' \mid K_d] \to P_i[G' \mid Q_{c',\widetilde{w}}]$.

   (d) A registration succeeds for a value $c' \neq c$. For some tuple of values $\widetilde{w}$ whose length is the arity of x and some $\mathsf{BitString}$ value c' that is not already in $\widetilde{u}$, we have $P_1[G' \mid Q_{c',\widetilde{w}}] \to\equiv C[R_x\{\widetilde{u}, c'\}G'] \mid x^\circ\langle\widetilde{w}\rangle]$. The process $P_2[G]$ can match this reduction, and the two resulting processes are still in $\mathcal{R}$ with $P \mid x^\circ\langle\widetilde{w}\rangle$ instead of P, with $\widetilde{u}, c'$ instead of $\widetilde{u}$, and with G' instead of G.

   (e) A registration succeeds for the value c: $G \equiv G' \mid Q_{c,\widetilde{w}}$ and by hypothesis on the occurrences of c in P_1 we obtain $\widetilde{w} = \widetilde{v}$, and thus $P_1' \equiv P_2[G']$. As described at the beginning of the proof, another $Q_{c,\widetilde{v}}$ can be obtained up to expansion, hence $P_1' \succeq P_2[G' \mid Q_{c,\widetilde{v}}] \equiv P_2[G]$. In this case only, no reduction on $P_2[G]$ is required to match the reduction on $P_1[G]$. □

We are now ready to prove all the correctness requirements stated in section 5 for the basic protocol.

Proof of correctness for the basic protocol: We use the following family $\mathbf{R}$ of receiving processes (up to structural equivalence):

$$\left\{R_x\{\widetilde{u}\}\left[\prod_{(c,\widetilde{v})\in S} Q_{c,\widetilde{v}}\right]\right\}$$

where $\{\widetilde{u}\}$ ranges over sets of values of type $\mathsf{BitString}$ in which x^- does not occur, and where S ranges over multisets of pairs $(c, \widetilde{v})$ such that the length of $\widetilde{v}$ is the arity of x and x^- does not occur in c or $\widetilde{v}$.

1: $R_x \equiv R_x\{\}[0]$.

2: This condition is syntactically obvious once we note that all the names of $\widetilde{u}$, $\widetilde{v}$, and c must be of type $\mathsf{BitString}$.

3: This condition is also syntactically obvious.

4: We must prove that

$$\mathcal{E}nv_{x^\pm}\,[\,R \mid E_x[\widetilde{v}]\,] \;\succeq\; \mathcal{E}nv_{x^\pm}\,[\,R \mid x^\circ\langle\widetilde{v}\rangle\,]$$

for every $R \in \mathbf{R}$.

By applying Lemma 5 with $X = \{emit, recv, x^+\}$, followed by structural equivalence, this statement is equivalent to: for all D and P such that $x^- \notin \mathsf{fv}[\mathsf{def}\ D\ \mathsf{in}\ P]$ and $x^-, x^+, emit, recv \notin \mathsf{dv}[D]$, for every $R \in \mathbf{R}$,

$$\begin{array}{rl} & \mathcal{E}_{x^\pm}\,[\,\mathsf{def}\ D\ \mathsf{in}\ P \mid W \mid R \mid E_x[\widetilde{v}]\,] \\ \succeq & \mathcal{E}_{x^\pm}\,[\,\mathsf{def}\ D\ \mathsf{in}\ P \mid W \mid R \mid x^\circ\langle\widetilde{v}\rangle\,] \end{array}$$

Next, we derive this expansion relation by composing those of Lemmas 12 and 11. For all S, $\{\widetilde{u}\}$, $\widetilde{v}$, D, and P in which x^- does not occur, for any fresh names c and d, we let $G = \prod_{(c',\widetilde{v}')\in S} Q_{c',\widetilde{v}'}$, and obtain the series of relations:

$$\begin{array}{rll} & \mathcal{E}_{x^\pm}\,[\,\mathsf{def}\ D\ \mathsf{in}\ P \mid W \mid R_x\{\widetilde{u}\}[G] \mid E_x[\widetilde{v}]\,] & \\ \equiv & \mathcal{E}_{x^\pm,c}\left[\,(\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid R_x\{\widetilde{u}\}[G])\left\{{}^{\{c,\widetilde{v}\}_{x^+}}/_{d}\right\}\right] & (1) \\ \succeq & \mathcal{E}_{x^\pm,c}\left[\,(\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid R_x\{\widetilde{u},c\}[G] \mid x^\circ\langle\widetilde{v}\rangle)\left\{{}^{\{c,\widetilde{v}\}_{x^+}}/_{d}\right\}\right] & (2) \\ \succeq & \mathcal{E}_{x^\pm,d}\,[\,\mathsf{def}\ D\ \mathsf{in}\ P \mid W_d \mid R_x\{\widetilde{u}\}[G] \mid x^\circ\langle\widetilde{v}\rangle\,] & (3) \\ \equiv & \mathcal{E}_{x^\pm}\,[\,\mathsf{def}\ D\ \mathsf{in}\ P \mid W \mid R_x\{\widetilde{u}\}[G] \mid x^\circ\langle\widetilde{v}\rangle\,] & (4) \end{array}$$

The first structural equivalence (1) is obtained by extending the scope of the new unique identifier c out of $E_x[\widetilde{v}] \stackrel{\text{def}}{=} \mathsf{def\ fresh}\ c\ \mathsf{in\ repl}\ emit\langle\{c,\widetilde{v}\}_{x^+}\rangle$; by hypothesis, the name c does not appear elsewhere in the initial process, thus there is no capture as its definition is lifted to $\mathcal{E}_{x^\pm,c}[\,\cdot\,]$. The expansion (2) is obtained by applying Lemma 12 for $n = 0$ and $G' = G$. The expansion (3) is obtained by applying Lemma 11, taking $P \mid x^\circ\langle\widetilde{v}\rangle$ for P and using that x^- does not occur in $\widetilde{v}$. The final structural equivalence (4) is obtained by first restricting the scope of d to the noise generator W_d—by hypothesis, d does not occur in any other part of the process—then by using the structural equivalence $W \equiv \mathsf{def\ fresh}\ d\ \mathsf{in}\ W_d$.

5: Suppose that

$$\mathcal{E}nv_{x^\pm}\,[\,R_x \mid emit\langle m\rangle\,] \rightarrow P$$

and that x^+ and x^- do not occur in m. The only possible reduction of $\mathcal{E}nv_{x^\pm}\,[\,R_x \mid emit\langle m\rangle\,]$ is the first reduction shown in Lemma 7, where the message $emit\langle m\rangle$ is consumed and turned into a message on a continuation channel, $\kappa\langle m\rangle$. Since x^+ and x^- do not occur in m, it follows that m cannot be a ciphertext with key x^+. Therefore, the case of Lemma 7 for failing decryptions applies, hence $P \succeq \mathcal{E}nv_{x^\pm}\,[\,R_x \mid emit\langle m\rangle\,]$.

6: We prove that if $R_x\{\widetilde{u}\}[G] \in \mathbf{R}$, x^- does not occur in m, and

$$\mathcal{E}nv_{x^\pm}\,[\,R_x\{\widetilde{u}\}[G] \mid emit\langle m\rangle\,] \to P$$

then

$$P \succeq \mathcal{E}nv_{x^\pm}\,\left[\,R' \mid T\,\right]$$

for some $R' \in \mathbf{R}$ and some process T such that x^- does not occur in T. The proof of this condition uses Lemma 7 too. It has several cases; the case analysis depends on the value of m and on the choice of reduction inside the receiving process.

1. Reception of a message $\{c, \widetilde{v}\}_{x^+}$ for some $\widetilde{v}$ such that its length equals the arity of x. Applying Lemma 7 we obtain

$$P \quad \succeq \quad \mathcal{E}nv_{x^\pm}\left[\,R_x\{\widetilde{u}\}[G \mid Q_{c,\widetilde{v}}]\,\right]$$

   Since x^- does not occur in $\widetilde{v}$, the pair $(c, \widetilde{v})$ can be added to the multiset S associated with G. We let $T = \mathsf{0}$ and $R' = R_x\{\widetilde{u}\}[G \mid Q_{c,\widetilde{v}}]$.

2. Reception of any other message m. Applying Lemma 7, we obtain

$$P \quad \succeq \quad \mathcal{E}nv_{x^\pm}\,[\,R_x\{\widetilde{u}\}[G] \mid emit\langle m\rangle\,]$$

3. Internal reduction (independently of the message). The only reductions in $R_x\{\widetilde{u}\}[G]$ occur when a process $Q_{c,\widetilde{v}}$ successfully registers the unique identifier c in the table of R_x, using rule Uid, and $G \equiv G' \mid Q_{c,\widetilde{v}}$ for some G'.

$$\begin{aligned} & \mathcal{E}nv_{x^\pm}\left[\,R_x\{\widetilde{u}\}\left[G' \mid Q_{c,\widetilde{v}}\right] \mid emit\langle m\rangle\,\right] \\ \to\ & \mathcal{E}nv_{x^\pm}\,[\,R_x\{\widetilde{u}, c\}\,[G' \mid x^\circ\langle\widetilde{v}\rangle] \mid emit\langle m\rangle\,] \end{aligned}$$

   We let $T = x^\circ\langle\widetilde{v}\rangle$. By the definition of $\mathbf{R}$, we have that x^- does not occur in $\widetilde{v}$ and that $R_x\{\widetilde{u}, c\}[G'] \in \mathbf{R}$. $\square$

B.4 Correctness of the session-based protocol

The proof for the session-based protocol has the same structure as the one for the basic protocol, but it is actually simpler. Although the session-based protocol has several steps, sessions are independent of one another; Lemmas 8 and 9 can be used to reason about each session separately. However, a technical complication is that a receiving process R_x may respond several times, and with several different keys, to messages from a single process $E_x[\widetilde{v}]$; until $E_x[\widetilde{v}]$ accepts a particular one of the keys, all these sessions are potentially useful.

In the proof, we use the following substitutions:

$$\begin{array}{lcll} \sigma_n & \stackrel{\text{def}}{=} & \left\{ {}^{\{k\}_{x^+}}/_{d}, {}^{\{k'_1\}_k}/_{d'_1}, \ldots, {}^{\{k'_n\}_k}/_{d'_n} \right\} & \text{for } n \geq 0 \\ \rho_n & \stackrel{\text{def}}{=} & \sigma_n \left\{ {}^{\{\widetilde{v}\}_{k'_n}}/_{d''} \right\} & \text{for } n \geq 1 \end{array}$$

We always assume that all the names d, d'_i, d'', k, k'_i are distinct. We apply these substitutions to a process only when the values substituted for d, d'_i, and d'' do not appear in the process before the substitution. When $\widetilde{v}$ and x are clear from context, we use also the following abbreviations for filtering receptions:

$$\begin{array}{lcl} H_k & \stackrel{\text{def}}{=} & \mathsf{let}\ k' = \mathsf{filter}\ k\ \mathsf{in}\ \mathsf{repl}\ emit\langle\{\widetilde{v}\}_{k'}\rangle \\ H'_{k'_i} & \stackrel{\text{def}}{=} & \mathsf{let}\ \widetilde{y} = \mathsf{filter}\ k'_i\ \mathsf{in}\ x^\circ\langle\widetilde{y}\rangle \\ Q_k & \stackrel{\text{def}}{=} & \mathsf{def\ key}\ k'\ \mathsf{in}\ (\mathsf{repl}\ emit\langle\{k'\}_k\rangle) \mid H'_{k'} \end{array}$$

In the arguments about the session-based protocol, we assume that G is the parallel composition of derivatives $K_{\langle m\rangle}$ or K_m of processes of the form $emit\langle m\rangle \mid F_x$, as described in Lemma 7. This convention is analogous to that for G in the basic protocol. In applying Lemma 7, here we take $Q = Q_k$.

Intuitively, Lemma 13 implies that all abortive sessions can be discarded (up to expansion) on the receiving side once the sending side has terminated.

Lemma 13 *For all $n \geq 1$, for all D and P, if k does not occur in D, P, and $\widetilde{v}$, if the names d'_i for $i > n$ and k'_i for $i \geq 1$ do not occur in D, P, and G, and if x^- occurs only as a decryption key in $R_x \mid G$, then we have:*

$$\begin{array}{rl} & \mathcal{E}_{x^\pm,k,k'_1,\ldots,k'_n}\left[\left(\mathsf{def}\ D\ \mathsf{in}\ R_x \mid G \mid P \mid W_{d,d'_1,\ldots,d'_n,d''} \mid \prod_{i=1}^{n-1} H'_{k'_i}\right)\rho_n \right] \\ \succeq & \mathcal{E}_{x^\pm,d,d'_1,\ldots,d'_n,d''}\left[\mathsf{def}\ D\ \mathsf{in}\ R_x \mid G \mid P \mid W_{d,d'_1,\ldots,d'_n,d''}\right] \end{array}$$

Proof: Let $\mathcal{R}$ be the relation that contains all pairs of processes (P_1, P_2) described in the statement of the lemma, for all $n \geq 1$. We use Lemma 3 with the set $\mathcal{N}' = \{x^+, x^-, d, d'_i \text{ for } i \geq 1, d'', k, k'_i \text{ for } i \geq 1, emit, recv\}$ to prove that $\mathcal{R} \subseteq \succeq$.

The relation $\mathcal{R}$ is closed by application of an evaluation context that does not use names in $\mathcal{N}'$ followed by structural rearrangement, so property (2) of Lemma 3 holds. Moreover, P_1 and P_2 have the same strong barbs, so property (1) holds. To verify the remaining properties, we consider the possible reductions $P_1 \to P_1'$ and $P_2 \to P_2'$ of P_1 and P_2. The two non-trivial cases are as follows.

1. In P_1, one of the $n-1$ filters $H'_{k'_i}$ receives a message from the network, as described in Lemma 7. The decryption that follows must fail because, by hypothesis, k'_i cannot be the encryption key of this message. Therefore, we have:

   $$P_1 \ \to \ P_1' \ \succeq \ P_1 \ \mathcal{R} \ P_2$$

   Thus, no reduction on P_2 is necessary in this case.

2. In P_1, the ciphertext $\{k\}_{x^+}$ is decrypted. Because of the hypothesis on the occurrences of x^-, this decryption must take place in G, so we must have $G \equiv G' \mid K_{\{k\}_{x^+}}$ for some G' of the same form as G. When the process $K_{\{k\}_{x^+}}$ performs the decryption, it becomes Q_k (up to strong equivalence). The key binder $\mathsf{key}\ k'$ in Q_k can be lifted to the network context after α-conversion of the key to k'_n, so $P_1' \sim P_1''$ where P_1'' is a process of the same shape as P_1 but with G' instead of G and with index $n+1$.

   On the other hand, the corresponding decryption process K_d in P_2 performs a failing-decryption reduction, and yields back a process K' structurally equivalent to $F_x \mid emit\langle d\rangle$. We can fold the process F_x into R_x, and fold the process $emit\langle d\rangle$ into $W_{d,\ldots}$. Hence, G can be replaced with G' in this context. Further, we can unfold a new replicated message from the noise generator, with the structural equivalence $W_{d,d'_1,\ldots,d'_n,d''} \equiv \mathsf{def}\ d'_{n+1}\ \mathsf{in}\ W_{d,d'_1,\ldots,d'_{n+1},d''}$. By applying this structural equivalence in P_2', we obtain a process P_2'' to which $\mathcal{R}$ relates P_1''. In short, we have:

   $$P_1 \to P_1' \ \sim \ P_1'' \quad \mathcal{R} \quad P_2'' \ \equiv \ P_2' \leftarrow P_2$$

The remaining reductions of P_1 and P_2 are in direct correspondence; they use the values $d\rho_n$, $d'_i\rho_n$, $d''\rho_n$ in P_1 instead of the values in d, d'_i, d'' in P_2. For each reduction, as in Lemma 6, we use that the substitution ρ_n acts bijectively on values and commutes with the reduction. $\square$

Lemma 14 states that the session-based protocol transmits messages reliably and secretly, and thus establishes correctness condition 4.

Lemma 14 *For all D, P, and $\widetilde{v}$ in which x^- does not occur free and such that the length of $\widetilde{v}$ is the arity of x, we have the expansion relation:*

$$\mathcal{E}_{x^\pm}\left[\mathsf{def}\ D\ \mathsf{in}\ R_x \mid E_x[\widetilde{v}] \mid P \mid W\right] \quad \succeq \quad \mathcal{E}_{x^\pm}\left[\mathsf{def}\ D\ \mathsf{in}\ R_x \mid x^\circ\langle\widetilde{v}\rangle \mid P \mid W\right]$$

Proof: We establish a more general result. Let k, d, d'_i, k'_i for $i \geq 1$ be distinct names of type $\mathsf{BitString}$, and let $\mathcal{R}$ be the relation such that:

$$\begin{array}{rl} & \mathcal{E}_{x^\pm,k,k'_1,\ldots,k'_n}\left[\left(\mathsf{def}\ D\ \mathsf{in}\ R_x \mid G \mid P \mid W_{d,d'_1,\ldots,d'_n} \mid H_k \mid \prod_{i=1}^n H'_{k'_i}\right)\sigma_n\right] \\ \mathcal{R} & \mathcal{E}_{x^\pm,d,d'_1,\ldots,d'_n}\left[\mathsf{def}\ D\ \mathsf{in}\ R_x \mid G \mid P \mid W_{d,d'_1,\ldots,d'_n} \mid x^\circ\langle\widetilde{v}\rangle\right] \end{array}$$

for every D, P, $n \geq 0$ such that d'_i for $i > n$, k, k'_i for $i \geq 1$, and d'' do not appear in D, P, $\widetilde{v}$, and G, such that d, d'_i for $i \geq 1$ do not occur in $\widetilde{v}$, and such that x^- occurs only in $R_x \mid G$. We write $P_{1,n}$ and $P_{2,n}$ as abbreviations for these two processes in the relation $\mathcal{R}$. We prove that $\mathcal{R} \subseteq \succeq$ using Lemma 3 with the same set of names as in the previous lemma ($\mathcal{N}' = \{x^+, x^-, d, d'_i$ for $i \geq 1, d''k, k'_i$ for $i \geq 1, \mathit{emit}, \mathit{recv}\}$). The statement of the lemma is structurally equivalent to $P_{1,0} \succeq P_{2,0}$ in the case $G = \mathsf{0}$.

We first check that $P_{1,n}(\rightarrow^*\succeq)^* P_{2,n}$. Intuitively, we iterate three times the sequence of reductions for successful decryption described in Lemma 7, for the three messages $\{k\}_{x^+}$, $\{k'_{n+1}\}_k$, and $\{\widetilde{v}\}_{k'_{n+1}}$. We obtain the relations:

$$\begin{array}{rcll}
P_{1,n} & \succeq & \mathcal{E}_{x^\pm,k,k'_1,\ldots,k'_n}\left[\left(\begin{array}{lll} & & R_x \mid G \mid P \mid Q_k \\ \mathsf{def}\ D\ \mathsf{in} & \mid & W_{d,d'_1,\ldots,d'_n} \\ & \mid & H_k \mid \prod_{i=1}^n H'_{k'_i}\end{array}\right)\sigma_{n+1}\right] & (1) \\
& \equiv & P_{1,n+1} & (2) \\
& \rightarrow\succeq\equiv & \mathcal{E}_{x^\pm,k,k'_1,\ldots,k'_{n+1}}\left[\left(\begin{array}{lll} & & R_x \mid G \mid P \\ \mathsf{def}\ D\ \mathsf{in} & \mid & W_{d,d'_1,\ldots,d'_{n+1},d''} \\ & \mid & \prod_{i=1}^{n+1} H'_{k'_i}\end{array}\right)\rho_{n+1}\right] & (3) \\
& \succeq & \mathcal{E}_{x^\pm,k,k'_1,\ldots,k'_{n+1}}\left[\left(\begin{array}{lll} & & R_x \mid G \mid P \mid x^\circ\langle\widetilde{v}\rangle \\ \mathsf{def}\ D\ \mathsf{in} & \mid & W_{d,d'_1,\ldots,d'_{n+1},d''} \\ & \mid & \prod_{i=1}^{n} H'_{k'_i}\end{array}\right)\rho_{n+1}\right] & (4) \\
& \succeq & \mathcal{E}_{x^\pm,d,d'_1,\ldots,d'_{n+1},d''}\left[\begin{array}{lll}\mathsf{def}\ D\ \mathsf{in} & & R_x \mid G \mid P \mid x^\circ\langle\widetilde{v}\rangle \\ & \mid & W_{d,d'_1,\ldots,d'_{n+1},d''}\end{array}\right] & (5) \\
& \equiv & P_{2,n+1} & (6) \\
& \equiv & P_{2,n} & (7)
\end{array}$$

The process $P_{1,n}$ contains as subprocesses the replicated emission $\mathsf{repl}\ \mathit{emit}\langle d\sigma_n\rangle$ (within $W_{d,\ldots}\sigma_n$) and the replicated filtering reception $\mathsf{repl}\ F_x$ (as R_x); thus the expansion (1) follows from Lemma 8. This expansion adds $Q_k \equiv \mathsf{def\ key}\ k'_{n+1}\ \mathsf{in}\ \mathsf{repl}\ emit\langle\{k'_{n+1}\}_k\rangle \mid H'_{k'_{n+1}}$.

By definition of $\mathcal{R}$, the name k'_{n+1} does not appear elsewhere in the process, so the binder $\mathsf{key}\ k'_{n+1}$ can be lifted to the network context using structural equivalence (2); we obtain $P_{1,n+1}$.

The next relation (3) is obtained by unfolding a message $emit\langle d'_{n+1}\rho_{n+1}\rangle$ from $(W_{d,d'_1,\ldots,d'_{n+1}})\rho_{n+1}$ and applying Lemma 7 to this message and to the filtering process H_k. This expansion starts a process $\mathsf{repl}\ emit\langle\{\widetilde{v}\}_{k'_{n+1}}\rangle$, which can also be written as $(\mathsf{repl}\ emit\langle d''\rangle)\rho_{n+1}$; putting this process in parallel with $(W_{d,d'_1,\ldots,d'_{n+1}})\rho_{n+1}$, we obtain $(W_{d,d'_1,\ldots,d'_{n+1},d''})\rho_{n+1}$.

The expansion (4) follows by Lemma 9: the process before the expansion contains as subprocesses the replicated emission $emit\langle d''\rho_{n+1}\rangle$ (within $(W_{\ldots,d''})\rho_{n+1}$) and the filtering reception $H'_{k'_{n+1}}$. The expansion replaces $H'_{k'_{n+1}}$ with $x^\circ\langle\widetilde{v}\rangle$.

The expansion (5) applies Lemma 13 with index $n+1$ to eliminate the processes H_{k_i}, taking $P \mid x^\circ\langle\widetilde{v}\rangle$ for P and using that x^- does not occur in $\widetilde{v}$.

The structural equivalence (6) eliminates the process $\mathsf{repl}\ emit\langle d''\rangle$. First, it restricts the scope of d'' to $W_{d,d'_1,\ldots,d'_{n+1},d''}$ only (since the name d'' cannot occur elsewhere). Then it uses the structural equivalence:

$$\mathsf{def\ fresh}\ d''\ \mathsf{in}\ W_{d,d'_1,\ldots,d'_{n+1},d''} \quad\equiv\quad W_{d,d'_1,\ldots,d'_{n+1}}$$

The final structural equivalence (7) is similar. It eliminates the process $\mathsf{repl}\ emit\langle d'_{n+1}\rangle$ (using the hypothesis about lack of occurrences of the name d'_{n+1} for $\mathcal{R}$ at rank n).

Since $\mathcal{R} \subseteq (\rightarrow^*\succeq)^*$, we use the special case of Lemma 3 to establish that $\mathcal{R} \subseteq\ \succeq$. Up to structural rearrangement, the relation $\mathcal{R}$ is closed by application of an evaluation context whose free names do not include names in $\mathcal{N}'$, so property (2) of Lemma 3 holds. In order to verify property (3), we consider the possible reductions $P_{1,n} \rightarrow P'_{1,n}$ and we argue that there exist corresponding reductions $P_{2,n} \rightarrow^= P'_{2,n}$. In the following two cases, the corresponding reductions $P_{2,n} \rightarrow^= P'_{2,n}$ are straightforward, and yield $P'_{1,n} \equiv\mathcal{R}\equiv P'_{2,n}$.

1. A reduction involves only D, P, $\mathcal{E}_{\ldots}[\,\cdot\,]$, and $W_{\ldots}$ (after application of the substitution σ_n).

2. $R_x \mid G$ receives a message, or decrypts a message other than $\{k\}_{x^+}$.

We use the relations described in Lemma 7 for analyzing other reductions; these pertain to the decryption of $\{k\}_{x^+}$ and to the join of a message with a reception offer issued by H_k or $H'_{k'_i}$.

3. $G\sigma_n$ decrypts the message $\{k\}_{x^+}$. On the other hand, G fails to decrypt d in P_2 and, up to strong equivalence, the resulting processes are in $\mathcal{R}$ at rank $n+1$:
$$P_{1,n} \to P'_{1,n} \sim P'_{1,n+1} \;\mathcal{R}\; P_{2,n+1} \leftarrow P_{2,n}$$

4. H_k or $H'_{k'_i}$ receives a message, but will fail to decrypt it. According to Lemma 7, the resulting process can revert to $P_{1,n}$ up to expansion. Hence, no reduction is needed in P_2:
$$P_{1,n} \to P'_{1,n} \succeq P_{1,n} \;\mathcal{R}\; P_{2,n}$$
(This case covers all receptions by $H'_{k'_i}$ because we have assumed that k'_i does not occur anywhere as an encryption key.)

5. H_k receives a message $\{u\}_k$. By hypothesis, k does not occur anywhere except possibly in H_k and in σ_n for $n > 0$, so $u = k'_i$ for some $i \in 1 \ldots n$ and $n > 0$. Using α-conversion, we can assume that $v = k'_n$ without loss of generality. The series of relations (3–6) described above says that $P_{1,n} \to P'_{1,n} \succeq P_{2,n}$. (The shift between $n+1$ and n is possible because $n > 0$.) $\square$

Proof of correctness for the session-based protocol: Because there is no shared state between sessions of the session-based protocol, we can let the set $\mathbf{R}$ of receiving processes be the singleton $\{R_x\}$.

1, 2, 3: These conditions are immediate.

4: This condition is an instance of Lemma 14 in the case where D is empty and $P = plug\langle emit, recv\rangle \mid plug_x\langle x^+\rangle$.

5: The process R_x has no internal reductions; whenever it receives a message, it attempts to decrypt it using x^-. If $\mathcal{E}nv_{x^\pm}[\,R_x \mid emit\langle v\rangle\,] \to P$ and x^+ and x^- do not occur in v, then v is not encrypted with x^+, so
$$P \;\;\succeq\;\; \mathcal{E}nv_{x^\pm}[\,R_x \mid emit\langle v\rangle\,]$$
follows from Lemma 7 (in the case where decryption fails).

6: This condition also follows from Lemma 7. The case where decryption fails is covered as in condition 5. When decryption succeeds, we have:
$$\mathcal{E}nv_{x^\pm}[\,R_x \mid emit\langle\{u\}_{x^+}\rangle\,] \;\;\to\succeq\;\; \mathcal{E}nv_{x^\pm}[\,R_x \mid Q_u\,]$$
and x^- does not occur in Q_u since it does not occur in u. $\square$

B.5 Soundness properties of the translations

Next, we show that both translations are equivalent, then we establish that they are fully abstract. Throughout this section, we assume a fixed, correct underlying communication protocol $(R_x, E_x[\,\cdot\,])$; we write $\mathbf{R}_x$ for a corresponding set of receiving processes, as described in Definition 1. We also let Σ be a fixed, finite set of types of the join-calculus that is closed under decomposition.

We make explicit and extend our assumptions on names by partitioning the names and by specifying some fixed functions $\cdot^+$, $\cdot^-$, and $\cdot^\circ$. For a set of names S, we write S^+, S^-, and S° for the sets of names obtained by decorating every name in S with a $+$, a $-$, or a $\circ$. We choose $\mathcal{T} = \bigcup_{\sigma\in\Sigma}\{t_\sigma, t'_\sigma\}$ such that names in $\mathcal{T}$ do not clash with the free names of R_x and $E_x[\,\cdot\,]$. Informally, $\mathcal{T}$ gathers the names that are private to the firewall. We distinguish two disjoint sets of names of the join-calculus $\mathcal{B}$ and $\mathcal{F}$ such that there is an infinite number of names of every type in Σ in both $\mathcal{B}$ and $\mathcal{F}$, and an infinite number of names of any sjoin-calculus type in $\mathcal{N} \setminus (\mathcal{B} \cup \mathcal{F})$. In the proofs of this section, we use elements of $\mathcal{B}$ for the names defined in a process P under consideration, and elements of $\mathcal{F}$ for the names free in P. We refer to the names in $\mathcal{B}$ as internal names and to the names in $\mathcal{F}$ as external names. We assume that the functions $\cdot^+$, $\cdot^-$, and $\cdot^\circ$ have the following properties.

1. The function $\cdot^+$ has domain $\mathcal{B} \cup \mathcal{F}$ and yields values of type $\mathsf{BitString}$.
2. The restriction of $\cdot^+$ to $\mathcal{B}$ is injective and yields names of type $\mathsf{BitString}$.
3. For every join-calculus type $\sigma \in \Sigma$, the restriction of $\cdot^+$ to names of type σ is a bijection with range $\{v \text{ of type } \mathsf{BitString} \mid \mathsf{fv}[v] \cap \mathcal{B}^- = \emptyset\}$.
4. The function $\cdot^\circ$ has domain $\mathcal{B}$, is injective, and maps every name $x \in \mathcal{B}$ to a name x° of type $\langle \mathsf{BitString}, \ldots, \mathsf{BitString}\rangle$ with the same arity as x.
5. The function $\cdot^-$ has domain $\mathcal{B}$, is injective, and yields names of type $\mathsf{BitString}$.
6. For all $x \in \mathcal{B}$, neither x^+ nor x^- is a free name of $E_{[\,\cdot\,]}[\,\cdot\,]$ or $R_{[\,\cdot\,]}$.

Because of these properties, the only free names of type $\mathsf{BitString}$ of a translation $[\![D]\!]$ or $[\![P]\!]$ are in the image of $\cdot^+$. We show this property by considering the ways in which free names of type $\mathsf{BitString}$ can appear in $[\![D]\!]$ or $[\![P]\!]$: (a) in the guards of comparisons; (b) as x^+ or in $\widetilde{v}^+$ as part of $E_x[\widetilde{v}^+]$; (c) as free names of $E_{[\,\cdot\,]}[\,\cdot\,]$ or $R_{[\,\cdot\,]}$. (Although x^+ and x^- can appear free in R_x, they are immediately bound in the surrounding context.) Trivially, cases (a) and (b) yield names in the image of $\cdot^+$. In case (c), the name cannot be in $\mathcal{B}^-$ (by assumption 6), so it is in the image of $\cdot^+$ (by assumption 3).

We first focus on the filtering translation. We introduce a shorter notation for the contents of firewalls: for all finite sets $X \subset \mathcal{B}$ and $F \subset \mathcal{F}$ we let:

$$D_{\Sigma(X,F)} \quad \stackrel{\text{def}}{=} \quad D_\Sigma(X \cup F) \wedge \bigwedge_{x \in X} D_{x^\circ \triangleright x} \wedge \bigwedge_{x \in F} D_{x \triangleright E_x}$$

For every type $\sigma \in \Sigma$, assumption 3 guarantees that the association table for σ in $D_\Sigma(X \cup F)$ is bijective.

The next lemma describes the unfolding of a firewall as a message crosses it, either from the context outside (intrusion of a message $z^\circ\langle\widetilde{y}^+\rangle$), or from the process inside (extrusion of a message $y\langle\widetilde{z}\rangle$).

Lemma 15 *For all finite sets of names $X \subset \mathcal{B}$ and $F \subset \mathcal{F}$, for every definition D, process P, and family of processes $R'_x \in \mathbf{R}_x$ for $x \in X$, if no decryption key $y^- \in \mathcal{B}^-$ occurs in D, P, $D_{\Sigma(X,F)}$, or R'_x for $x \neq y$, and if names in $\mathcal{T}$ occur only in $D_{\Sigma(X,F)}$, then we have the expansion relations:*

1. *for all well-formed messages $z\langle\widetilde{y}\rangle$ with $z \in X$ and $\{\widetilde{y}\} \subset X \cup \mathcal{F}$, for $Y \stackrel{def}{=} \{\widetilde{y}\} \setminus (X \cup F)$,*

$$\begin{array}{rl} & \mathcal{E}_{X^\pm}\left[\mathsf{def}\ D_{\Sigma(X,F)} \wedge D\ \mathsf{in}\ z^\circ\langle\widetilde{y}^+\rangle \mid \prod_{x \in X} R'_x \mid P\right] \\ \succeq & \mathcal{E}_{X^\pm}\left[\mathsf{def}\ D_{\Sigma(X,F \cup Y)} \wedge D\ \mathsf{in}\ z\langle\widetilde{y}\rangle \mid \prod_{x \in X} R'_x \mid P\right] \end{array}$$

2. *for all well-formed messages $y\langle\widetilde{z}\rangle$ with $y \in F$ and $\{\widetilde{z}\} \subset \mathcal{B} \cup F$, for $Z \stackrel{def}{=} \{\widetilde{z}\} \setminus (X \cup F)$,*

$$\begin{array}{rl} & \mathcal{E}_{X^\pm}\left[\mathsf{def}\ D_{\Sigma(X,F)} \wedge D\ \mathsf{in}\ y\langle\widetilde{z}\rangle \mid \prod_{x \in X} R'_x \mid P\right] \\ \succeq & \mathcal{E}_{(X \cup Z)^\pm}\left[\mathsf{def}\ D_{\Sigma(X \cup Z,F)} \wedge D\ \mathsf{in}\ E_y[\widetilde{z}^+] \mid \prod_{x \in X} R'_x \mid \prod_{z \in Z} R_z \mid P\right] \end{array}$$

Proof: The traversal of the firewall and its recursive unfolding are performed by the series of reductions that are informally described in section 4.2. Some of these reductions are deterministic; Lemma 4 accounts for the others. Therefore, every reduction can be replaced with an expansion, and we conclude by transitivity. □

In what follows, we use normal forms for processes and their translations. For any join-calculus process P, we have $P \equiv M' \mid T$ or $P \equiv \mathsf{def}\ D\ \mathsf{in}\ M \mid M' \mid T$, where M is a parallel composition of messages sent on names defined in D,

M' is a parallel composition of messages sent on free names, and T is a parallel composition of processes of the form $\mathsf{if}\ v = w\ \mathsf{then}\ Q\ \mathsf{else}\ Q'$. Each of M, M', and T may be $\mathsf{0}$. We typically consider only the case $\mathsf{def}\ D\ \mathsf{in}\ M \mid M' \mid T$, since the case $M' \mid T$ is analogous but simpler.

The next lemma relates the compositional translation to a variant where the cryptographic protocol is short-circuited for messages on channels with internal names. We rely on an auxiliary translation $[\![\,\cdot\,]\!]^\circ$ with the same structural definition as the compositional translation $[\![\,\cdot\,]\!]$ of section 4 except for messages $x\langle\widetilde{v}\rangle$ in evaluation contexts:

$$[\![x\langle\widetilde{v}\rangle]\!]^\circ \quad \stackrel{\text{def}}{=} \quad \begin{cases} x^\circ\langle\widetilde{v}^+\rangle & \text{when } x \in \mathcal{B} \\ [\![x\langle\widetilde{v}\rangle]\!] & \text{otherwise} \end{cases}$$

Lemma 16 *Let $X \subset \mathcal{B}$ and $F \subset \mathcal{F}$ be two finite sets of names, let $C[\,\cdot\,]$ be a context of the form:*

$$C[\,\cdot\,] \quad \stackrel{\textit{def}}{=} \quad \mathcal{E}_{X^\pm}\left[\mathsf{def}\ D'\ \mathsf{in}\ [\,\cdot\,] \mid Q \mid W \mid \prod_{x \in X} R'_x\right]$$

where $R'_x \in \mathbf{R}_x$ for every $x \in X$, and where the names in $\mathcal{B}^-$, $(\mathcal{B} \setminus X)^\circ$, and $(\mathcal{B} \setminus X)^+$ are not free in D', Q, F^+, or R'_x (except for x°, x^+, and x^- in R'_x), and let P be a join-calculus process with $\mathsf{fv}[P] \subseteq X \cup F$ and with bound names in $\mathcal{B} \setminus X$. We have the expansion relation:

$$C[[\![P]\!]] \;\succeq\; C[[\![P]\!]^\circ]$$

Proof: Let us assume that $P \equiv \mathsf{def}\ D\ \mathsf{in}\ M \mid M_F \mid T$ where M and M_F are parallel compositions of messages, where T is a parallel composition of name comparisons, and where $\mathsf{dv}[D]$ is a subset of $\mathcal{B} \setminus X$ such that the names in $\mathsf{dv}[D]$, $\mathsf{dv}[D]^\circ$, $\mathsf{dv}[D]^+$, and $\mathsf{dv}[D]^-$ do not appear anywhere in $C[\,\cdot\,]$. By hypothesis, $\mathsf{fv}[M \mid M_F] \subseteq \mathsf{dv}[D] \cup X \cup F$, and we further assume that all messages in M are sent on names in $\mathsf{dv}[D] \cup X$ and all messages in M_F are sent on names in F. We let:

$$P_1 \;\stackrel{\text{def}}{=}\; \mathcal{E}_{(X \uplus \mathsf{dv}[D])^\pm}\left[\mathsf{def}\ D' \wedge [\![D]\!]\ \mathsf{in}\ [\![M]\!] \;\mid [\![M_F \mid T]\!] \mid Q \mid W \mid \prod_{x \in X} R'_x \mid \prod_{x \in \mathsf{dv}[D]} R_x\right]$$

$$P_2 \;\stackrel{\text{def}}{=}\; \mathcal{E}_{(X \uplus \mathsf{dv}[D])^\pm}\left[\mathsf{def}\ D' \wedge [\![D]\!]\ \mathsf{in}\ [\![M]\!]^\circ \mid [\![M_F \mid T]\!] \mid Q \mid W \mid \prod_{x \in X} R'_x \mid \prod_{x \in \mathsf{dv}[D]} R_x\right]$$

Since by hypothesis the names in $\mathsf{dv}[[\![D]\!]] \cup \mathsf{dv}[D]^+ \cup \mathsf{dv}[D]^-$ do not occur in $C[\,\cdot\,]$ or in F^+, we have the structural equivalences $C[[\![P]\!]] \equiv P_1$ and $C[[\![P]\!]^\circ] \equiv P_2$.

By construction, the translations $[\![\cdot]\!]$ and $[\![\cdot]\!]^\circ$ differ only on the messages $x\langle\widetilde{y}\rangle$ in M, for which there is a receiving process R'_x or R_x. We repeatedly apply correctness condition 4 for every such message. We assume that $M \equiv M' \mid x\langle\widetilde{y}\rangle$, and we let:

$$\begin{array}{rcl} D'' & \stackrel{\text{def}}{=} & D' \wedge [\![D]\!] \wedge \bigwedge_{z\in(X\cup\mathsf{dv}[D])\setminus\{x\}}\mathsf{keys}\ z^+, z^- \\ P'' & \stackrel{\text{def}}{=} & [\![M']\!]^\circ \mid [\![M_F \mid T]\!] \mid Q \mid \prod_{z\in X\setminus\{x\}} R'_z \mid \prod_{z\in\mathsf{dv}[D]\setminus\{x\}} R_z \end{array}$$

Since $x \in X \cup \mathsf{dv}[D]$, since $X \cup \mathsf{dv}[D] \subset \mathcal{B}$, since the names $\mathcal{B}^-$ do not occur in D' or Q, since x^- does not appear in R'_y (by hypothesis) or in R_y (by assumption 6) for $x \neq y$, and since the only free names of type $\mathsf{BitString}$ of $[\![D]\!]$, $[\![M]\!]$ (and hence $[\![M']\!]^\circ$), and $[\![M_F \mid T]\!]$ are in the image of $\cdot^+$, we obtain that x^- does not occur anywhere in P'' or D'' (by assumption 3), so we can apply correctness condition 4 to $E_x[\widetilde{y}^+]$ and R'_x, and then Lemma 5 with D'' and P''. (The correctness condition requires that x^- does not occur in $\widetilde{y}^+$; this holds by assumption 3.) We obtain the expansion relation:

$$\begin{array}{rl} & \mathcal{E}_{x^\pm}\left[\,\mathsf{def}\ D'' \ \mathsf{in}\ P'' \mid W \mid R'_x \mid [\![x\langle\widetilde{y}\rangle]\!]\,\right] \\ \succeq & \mathcal{E}_{x^\pm}\left[\,\mathsf{def}\ D'' \ \mathsf{in}\ P'' \mid W \mid R'_x \mid [\![x\langle\widetilde{y}\rangle]\!]^\circ\,\right] \end{array}$$

By composing the expansion relations for all the messages in M, we obtain $P_1 \succeq P_2$, hence $C[[\![P]\!]] \succeq C[[\![P]\!]^\circ]$. □

Our next lemma relates the compositional translation and the filtering translation of the same join-calculus process, in a context that defines the network and features some low-level definitions D_A and processes P_A. The processes considered are in a stable form up to expansion (that is, they are in a form that reflects possible communication with the context, and that is not broken by subsequent communication with the context followed by applications of the expansion relations of the previous lemmas).

Lemma 17 *For every join-calculus process* $\mathsf{def}\ D\ \mathsf{in}\ P$*, sjoin-calculus process* $\mathsf{def}\ D_A\ \mathsf{in}\ P_A$*, finite sets of names X, L, and F, and family of processes $R'_x \in \mathbf{R}_x$ indexed by $x \in X$, we have the observational equivalence:*

$$\begin{array}{rl} & \mathcal{E}_{(X\cup L)^\pm}\left[\mathsf{def}\ D_A \wedge [\![D]\!]\ \mathsf{in}\ P_A \mid [\![P]\!]^\circ \mid W \mid \prod_{x\in X} R'_x \mid \prod_{x\in L} R_x\right] \\ \approx & \mathcal{E}_{X^\pm}\left[\mathsf{def}\ D_A \wedge D_{\Sigma(X,F)} \wedge D\ \mathsf{in}\ P_A \mid P \mid W \mid \prod_{x\in X} R'_x\right] \end{array}$$

provided that:

1. $\mathsf{dv}[D] = X \uplus L \subset \mathcal{B}$ *and* $\mathsf{fv}[\mathsf{def}\ D\ \mathsf{in}\ P] \subseteq F \subset \mathcal{F}$.

2. *Names in $\mathcal{B} \uplus \mathcal{F}$ occur only in D, P, and $D_{\Sigma(X,F)}$.*

3. *Names x° in $(\mathcal{B} \setminus X)^\circ$ occur only in $[\![D]\!]$ and R_x, and $X^\circ \cap \mathsf{dv}[D_A] = \emptyset$.*

4. *Names x^+ in $(\mathcal{B} \setminus X)^+$ occur only in $[\![P]\!]^\circ$, $[\![D]\!]$, and R_x (but not in $\mathsf{fv}[F^+]$).*

5. *Names x^- in $\mathcal{B}^-$ occur only in R'_x and R_x.*

6. *Names in $\mathcal{T}$ occur only in $D_{\Sigma(X,F)}$.*

7. *Names bound in P and in every process within D are all in $\mathcal{B} \setminus X$.*

Proof: Let $\mathcal{N}' = \mathcal{B} \uplus \mathcal{B}^\circ \uplus \mathcal{B}^+ \uplus \mathcal{B}^- \uplus \mathcal{F} \uplus \mathcal{T} \uplus \{emit, recv\}$, let $\mathcal{R}$ be the relation that contains all the pairs of processes (P_1, P_2) that meet the invariants of the lemma, and let $\mathcal{R}' \subseteq \mathcal{R}$ be the subrelation whose processes meet the additional invariant:

8. *P is $M \mid T$ where M is a parallel composition of messages $x\langle\widetilde{v}\rangle$ with $x \in \mathsf{dv}[D]$ and where T is a parallel composition of comparisons.*

We first prove that $\mathcal{R} \subseteq \equiv\mathcal{R}'\preceq$ by showing how to normalize a pair of processes (P_1, P_2) in $\mathcal{R}$ to a pair of processes in $\mathcal{R}'$. To enforce invariant 8, we use a normal form $P \equiv \mathsf{def}\ D'\ \mathsf{in}\ M \mid M' \mid T$ where $\mathsf{dv}[D'] \subset \mathcal{B} \setminus \mathsf{dv}[D]$ (by α-conversion) and where M and M' are parallel compositions of messages sent on names in $X \cup L \cup \mathsf{dv}[D']$ and in F, respectively.

In P_1, our hypotheses guarantee that the structural equivalence above commutes with the auxiliary translation: for every name $x \in \mathcal{B} \setminus (X \cup L)$ introduced by α-conversion, the invariants 3, 4, and 5, respectively, ensure that the names x°, x^+, and x^- obtained from our functions on names do not occur in P_1 in the same scope as $[\![P]\!]^\circ$. We apply α-conversion in P_1 to use these particular names, unfold the definition of $[\![\mathsf{def}\ D'\ \mathsf{in}\ M \mid M' \mid T]\!]^\circ$, then lift the definitions $[\![D']\!]$ and $\wedge_{x \in \mathsf{dv}[D']}\ \mathsf{keys}\ x^+, x^-$ that appear in the translation to put them in conjunction with D and with the key definitions in $\mathcal{E}_{X \cup L}[\ \cdot\ ]$, respectively.

In P_2, each message $y\langle\widetilde{z}\rangle$ within M' is such that $y \in F$, hence we can apply Lemma 15(2) to unfold the firewall; as a result, the tables in the firewall are extended with pairs (z, z^+) for each name $z \in \mathsf{fv}[M'] \setminus (X \cup F)$, while the message $y\langle\widetilde{z}\rangle$ is replaced with the emitting process $[\![y\langle\widetilde{z}\rangle]\!] = E_y[\widetilde{z}^+]$.

Applying structural equivalence, we have $P_1 \equiv P'_1\ \mathcal{R}\ P'_2 \preceq\equiv P_2$ for a pair of processes P'_1 and P'_2 derived from P_1 and P_2 by substituting

- $X' \stackrel{\text{def}}{=} X \cup (\mathsf{fv}[M'] \setminus F)$ for X,
- $L' \stackrel{\text{def}}{=} (L \cup \mathsf{dv}[D']) \setminus \mathsf{fv}[M']$ for L,
- $D \wedge D'$ for D,

- $P_A \mid [\![M']\!]$ for P_A,
- $M \mid T$ for P.

We easily check that P'_1 and P'_2 meet all eight invariants:

1. Let $U \stackrel{\text{def}}{=} \mathsf{dv}[D'] \cap \mathsf{fv}[M']$ and $V \stackrel{\text{def}}{=} L \cap \mathsf{fv}[M']$. Decomposing $\mathsf{fv}[M']$ according to the partition $X \uplus \mathsf{dv}[D'] \uplus L \uplus F$, we obtain $X' = X \cup (U \uplus V)$ and $L' = (L \cup \mathsf{dv}[D']) \setminus (U \uplus V)$; moreover, the sets X, U, $\mathsf{dv}[D'] \setminus U$, V, and $L \setminus V$ are disjoint subsets of $\mathcal{B}$. We rearrange names defined in $D \wedge D'$ to meet the new instance of the invariant:

$$\begin{aligned}
\mathsf{dv}[D \wedge D'] &= (X \uplus L) \uplus \mathsf{dv}[D'] \\
&= (X \uplus (L \setminus V) \uplus V) \uplus (U \uplus (\mathsf{dv}[D'] \setminus U)) \\
&= (X \uplus U \uplus V) \uplus ((L \uplus \mathsf{dv}[D']) \setminus (U \uplus V)) \\
&= X' \uplus L'
\end{aligned}$$

2. We check that $\mathsf{fv}[[\![M']\!]] \cap (\mathcal{B} \uplus \mathcal{F}) = \emptyset$.

3. Additional names in $\mathcal{B}^\circ$ appear only in $\mathsf{dv}[[\![D']\!]] = \mathsf{dv}[D']^\circ$.

4. Additional names in $\mathcal{B}^+$ appear for $x \in \mathsf{dv}[D']$ in translated terms and processes R'_x; they appear in $[\![M']\!]$ only when they also appear in $X' \setminus X$.

5. Additional names in $\mathcal{B}^-$ appear only in R'_x and R_x for $x \in \mathsf{dv}[D']$.

6. These names are not affected.

7. Additional bound names appear only as we choose a normal form for P.

8. This holds by construction of M and T.

Next, we study the names that may occur free in processes (P_1, P_2) related by $\mathcal{R}$, and argue that these names are not in $\mathcal{N}'$. Within P_1, the context $\mathcal{E}_{(X \cup L)^\pm}[\,\cdot\,]$ binds $(X \cup L)^+ \cup (X \cup L)^- \cup \{emit, recv\}$ and, by definition of $[\![D]\!]$ plus invariant 1, we have $(X \cup L)^\circ = \mathsf{dv}[[\![D]\!]]$, so

$$\begin{aligned}
\mathsf{fv}[P_1] \;\subseteq\; & (\mathsf{fv}[\mathsf{def}\ D_A\ \mathsf{in}\ P_A] \cup \mathsf{fv}[[\![D]\!]] \cup \mathsf{fv}[[\![P]\!]^\circ] \cup \mathsf{fv}[W] \cup \mathsf{fv}[\prod_{x \in X} R'_x \mid \prod_{x \in L} R_x]) \\
\setminus\; & ((X \cup L)^+ \cup (X \cup L)^- \cup (X \cup L)^\circ \cup \{emit, recv\})
\end{aligned}$$

Within P_2, the context $\mathcal{E}_{X^\pm}[\,\cdot\,]$ binds $X^+ \cup X^- \cup \{emit, recv\}$, and we have $X^\circ \cup F \cup \mathcal{T} \subseteq \mathsf{dv}[D_{\Sigma(X,F)}]$ by definition of $D_{\Sigma(X,F)}$ and $X \cup L = \mathsf{dv}[D]$ by invariant 1, so

$$\begin{aligned}
\mathsf{fv}[P_2] \;\subseteq\; & (\mathsf{fv}[\mathsf{def}\ D_A\ \mathsf{in}\ P_A] \cup \mathsf{fv}[D] \cup \mathsf{fv}[D_{\Sigma(X,F)}] \cup \mathsf{fv}[P] \cup \mathsf{fv}[W] \cup \mathsf{fv}[\prod_{x \in X} R'_x]) \\
\setminus\; & (X \cup L \cup F \cup X^+ \cup X^- \cup X^\circ \cup \mathcal{T} \cup \{emit, recv\})
\end{aligned}$$

We now detail the free variables of the subterms above. We have $\mathsf{fv}[W] = \{emit\}$. By invariants 2–6, $(\mathsf{fv}[D_A] \cup \mathsf{fv}[P_A]) \cap \mathcal{N}' \subseteq X^+ \cup X^\circ \cup \{emit, recv\}$. By correctness condition 2 on the protocol, $\mathsf{fv}[E_x[\,\cdot\,]] \cup \mathsf{fv}[R'_x] \cup \mathsf{fv}[R_x]$ with $x \in X \cup L$ may contain names in $\{emit, recv, x^\circ, x^+, x^-\}$ bound in the enclosing context, and other names of type $\mathsf{BitString}$, which must be outside $\mathcal{N}'$ by assumption 6 on $\cdot^+$ and $\cdot^-$. We let Z be the set of these other names. By invariant 1, $\mathsf{fv}[D] \cup \mathsf{fv}[P] \subseteq X \cup L \cup F$. By definition of the translation, $\mathsf{fv}[[\![D]\!]] \cup \mathsf{fv}[[\![P]\!]^\circ] \subseteq (X \cup L)^+ \cup (X \cup L)^- \cup (X \cup L)^\circ \cup \mathsf{fv}[F^+] \cup \{emit, recv\} \cup Z$. By definition of the firewall, $\mathsf{fv}[D_{\Sigma(X,F)}] \subseteq \mathsf{dv}[D_{\Sigma(X,F)}] \cup X \cup X^+ \cup X^- \cup \mathsf{fv}[F^+] \cup \{emit, recv\} \cup Z$. Next, we check that names in $\mathsf{fv}[F^+] \cap \mathcal{N}'$ cannot be free in P_1 or P_2: since names in $\mathsf{fv}[F^+]$ are of type $\mathsf{BitString}$, $\mathsf{fv}[F^+] \cap \mathcal{N}' \subseteq \mathcal{B}^+ \cup \mathcal{B}^-$; by invariant 4, names x^+ in $\mathsf{fv}[F^+] \cap \mathcal{B}^+$ are limited to names in X^+, which are bound in the context; by assumption 3 on $\cdot^+$, names in $\mathcal{B}^-$ are excluded. In summary, we obtain:

$$\mathsf{fv}[P_1] \cup \mathsf{fv}[P_2] \quad \subseteq \quad (\mathsf{fv}[\mathsf{def}\ D_A\ \mathsf{in}\ P_A] \cup \mathsf{fv}[F^+] \cup Z) \setminus \mathcal{N}'$$

We now use Lemma 2 to prove that $\mathcal{R}' \subseteq \approx$. We let $P_1\ \mathcal{R}'\ P_2$ and argue that P_1 and P_2 satisfy the five properties required in Lemma 2. Property 5 of Lemma 2 immediately follows from our analysis of $\mathsf{fv}[P_1] \cup \mathsf{fv}[P_2]$. To establish property 1, we remark that the strong barbs of P_1 and P_2 coincide by arguing that they are exactly the strong barbs of P_A on channel names outside $\mathcal{N}'$ that are not bound by D_A.

To establish property 2, we consider an evaluation context of the form $\mathsf{def}\ D_B\ \mathsf{in}\ P_B \mid [\cdot]$ such that $\mathsf{fv}[\mathsf{def}\ D_B\ \mathsf{in}\ P_B] \cap \mathcal{N}' = \emptyset$. Using α-conversion after applying this context to P_1 and P_2, we can assume that $\mathsf{dv}[D_A] \cap (\mathsf{fv}[D_B] \cup \mathsf{fv}[P_B]) = \emptyset$ and $\mathsf{dv}[D_B] \cap \mathcal{N}' = \emptyset$. Then, we have $\mathsf{def}\ D_B\ \mathsf{in}\ P_B \mid P_1 \equiv P'_1\ \mathcal{R}'\ P'_2 \equiv \mathsf{def}\ D_B\ \mathsf{in}\ P_B \mid P_2$ for the processes P'_1 and P'_2 obtained from P_1 and P_2 by substituting $D_A \wedge D_B$ for D_B and $P_A \mid P_B$ for P_A. Invariants 1, 7, and 8 are not affected; invariants 2–6 immediately follow from $(\mathsf{fv}[D_B] \cup \mathsf{fv}[P_B]) \cap \mathcal{N}' = \emptyset$.

To establish the bisimulation properties (3 and 4), we prove that, for every reduction $P_1 \rightarrow Q_1$, there are processes P'_1 and P'_2 such that

$$P_1 \ \rightarrow\ Q_1 \ \succeq\ P'_1\ \mathcal{R}\ P'_2\ \preceq^* \leftarrow^*\ P_2$$

and vice versa for every reduction $P_2 \rightarrow Q_2$. (Since $\mathcal{R} \subseteq {\equiv}\mathcal{R}'{\preceq}$, this suffices to obtain the corresponding relations with $\mathcal{R}'$ instead of $\mathcal{R}$, as required in Lemma 2.) The proof is by cases, relying on the following partition of reductions: reductions internal to D_A and P_A, reductions using the join-pattern modelling the network in $\mathcal{E}[\,\cdot\,]$, reductions internal to a particular receiving process in $\prod_{x \in X} R'_x$ or $\prod_{x \in L} R_x$, reductions using a join-pattern of D or its translation $[\![D]\!]$, reductions of comparisons in T or its translation $[\![T]\!]$, and reductions using a join-pattern in $D_{\Sigma(X,F)}$.

Reductions internal to D_A and P_A: Every reduction internal to P_A or of a join-pattern of D_A matched in P_A is the same on both sides of $\mathcal{R}'$; the processes P'_1 and P'_2 are obtained from P_1 and P_2 by substituting new terms D'_A and P'_A for D_A and P_A, with at most the same free names.

Reductions using the join-pattern of $\mathcal{E}[\,\cdot\,]$ and reductions internal to a receiving process R'_x: In case of a reduction using $\mathcal{E}[\,\cdot\,]$, the emission $emit\langle m \rangle$ is necessarily in P_A or W (not in R_x or R'_x, by correctness condition 3) and thus available both in P_1 and P_2; besides, we can assume that the emission is in P_A only, up to structural equivalence: when m is defined in $W \equiv W \mid \mathsf{def}$ fresh m in repl $emit\langle m \rangle$, the name m can be chosen outside $\mathcal{N}'$, the definition fresh m is added to D_A, and the replicated process repl $emit\langle m \rangle$ is added to P_A. The reception request may come from P_A or from a receiving process, R_x or R'_x.

- The case where the receiving process is in P_A is analogous to those for internal reductions of P_A.
- When the receiving process is R_x for some $x \in L$ (in P_1 only), the message $emit\langle m \rangle$ is consumed and the process R_x is replaced with some other process R in Q_1. The reduction uses only the network definition in the context, hence we also have

  $$\mathcal{E}nv_{x^\pm}[\,R_x \mid emit\langle m \rangle\,] \quad \rightarrow \quad \mathcal{E}nv_{x^\pm}[\,R\,]$$

  for any choice of fresh names $plug$, $plug_x$. By hypothesis, the keys x^+, x^- do not occur in m (since by invariants 4 and 5 they do not occur in P_A or W), so correctness condition 5 implies the expansion relation:

  $$\mathcal{E}nv_{x^\pm}[\,R\,] \quad \succeq \quad \mathcal{E}nv_{x^\pm}[\,R_x \mid emit\langle m \rangle\,]$$

  By applying Lemma 5 to this expansion, we show that R can be replaced with $R_x \mid emit\langle m \rangle$ also in Q_1 up to expansion, that is, $Q_1 \succeq P_1$. To apply the lemma, we choose D'_A and P'_A such that def D_A in $P_A \mid W \equiv$ def D'_A in $P'_A \mid emit\langle m \rangle \mid W$, and use the following D'' and P'' (D and P in Lemma 5):

  $$\begin{array}{rcl} D'' & \stackrel{\text{def}}{=} & \bigwedge_{z \in X \cup L \setminus \{x\}} \mathsf{keys}\ z^+, z^- \wedge D'_A \wedge [\![D]\!] \\ P'' & \stackrel{\text{def}}{=} & P'_A \mid M^\circ \mid [\![T]\!] \mid \prod_{z \in X} R'_z \mid \prod_{z \in L \setminus \{x\}} R_z \end{array}$$

  In this case, no reduction on P_2 is needed—we can take $P'_2 \equiv P_2$.

- When the receiving process is R'_x in $\mathbf{R}_x$ for some x in X (in either P_1 or P_2), by hypothesis the decryption key x^- does not occur in m (since by invariant 5 it does not occur in P_A or W). Since the reduction involves only these two processes and the join-pattern $emit\langle m\rangle \mid recv\langle\kappa\rangle$ in $\mathcal{E}[\,\cdot\,]$, it is enabled in both P_1 and P_2, and also corresponds to a reduction

$$\mathcal{E}nv_{x^\pm}\left[\, R'_x \mid emit\langle m\rangle \,\right] \quad\rightarrow\quad \mathcal{E}nv_{x^\pm}\left[\, R \,\right]$$

for any choice of fresh names $plug$, $plug_x$. By correctness condition 6, we obtain the expansion relation

$$\mathcal{E}nv_{x^\pm}\left[\, R \,\right] \quad\succeq\quad \mathcal{E}nv_{x^\pm}\left[\, R''_x \mid Q'' \,\right]$$

for some $R''_x \in \mathbf{R}_x$ and some process Q'' with at most the same free names as in $emit\langle m\rangle$ and R'_x minus x^-.

We apply Lemma 5 twice to derive two other expansions $Q_1 \succeq P'_1$ and $Q_2 \succeq P'_2$. (We rely on expressions D'' and P'' similar to those of the previous case.) We obtain the relations

$$P_1 \rightarrow Q_1 \succeq P'_1 \equiv\mathcal{R}\equiv P'_2 \preceq Q_2 \leftarrow P_2$$

where structural equivalence is used to rearrange the common parts of P'_1 and P'_2. The new related processes differ from P_1 and P_2 on the following points: D_A and P_A may have been rearranged up to structural equivalence; the process Q'' has been added to P_A; in case $emit\langle m\rangle$ was within W, the two components of a replicated emission $\mathsf{def\ fresh}\ m\ \mathsf{in\ repl}\ emit\langle m\rangle$ have been added to D_A and P_A respectively; otherwise the message $emit\langle m\rangle$ has been removed from P_A.

The case of a process R'_x performing an internal reduction (either in P_1 or P_2), instead of consuming a message $emit\langle m\rangle$, is handled in the same manner, as a special case. (Formally, there are always pending messages $emit\langle m\rangle$ issued by the noise generator; any of these messages can be used as described above, is left unaffected by the reduction, and remains in Q''.)

Reductions in $[\![D]\!]$ and D: In case the reduction uses a reaction rule in $[\![D]\!]$ or in D, respectively, this reduction is associated with a particular reduction that uses some rule $J \triangleright U$ in D to consume the messages $J\sigma$ and trigger the process $U\sigma$. We analyze first reductions in P_1 (relying on translated terms), then in P_2 (relying on filters instead).

- A reduction $P_1 \rightarrow Q_1$ uses the join-pattern of the translation of some rule $J \triangleright U$ in D to consume the process $[\![J]\!]^\circ\sigma$ and trigger the process

$[\![U]\!]\sigma$. By definition of $\mathcal{R}'$, the process $[\![J]\!]^\circ\sigma$ is a parallel composition of messages that appear in $[\![M]\!]^\circ$ and P_A. We rearrange M and P_A as follows: we let $M \equiv M' \mid M''$ and $P_A \equiv \mathsf{def}\ D'_A\ \mathsf{in}\ M'_A \mid P'_A$ such that $[\![J]\!]^\circ\sigma \equiv [\![M']\!]^\circ \mid M'_A$ and $\mathsf{dv}[D'_A] \cap \mathcal{N}' = \emptyset$.

After the reduction occurs, we show how to obtain a process of the shape required in the definition of $\mathcal{R}$. Every message that appears in M'_A is of the form $x^\circ\langle\widetilde{v}\rangle$ with $x \in X$. Suppose that x has type $\langle\widetilde{\tau}\rangle$; by assumption 3 on $\cdot^+$, there is a tuple of join-calculus names $\widetilde{y}$ in $X \uplus \mathcal{F}$ of types $\widetilde{\tau}$ such that $\widetilde{y}^+ = \widetilde{v}$. We augment F with the names in $\widetilde{y}$ that are not already present in X or F.

In Q_1, the triggered process is $[\![U]\!]\sigma$. We build a substitution σ' that operates on $\mathsf{rv}[J]$ and such that $[\![U\sigma']\!] = [\![U]\!]\sigma$. Every message $x^\circ\langle\widetilde{y}^+\rangle$ consumed in the reduction is matched by a message pattern $x^\circ\langle\widetilde{z}^+\rangle$ of $[\![J]\!]$ and σ is the substitution obtained by composing the substitutions $\{^{\widetilde{y}^+}/_{\widetilde{z}^+}\}$ for all x in $\mathsf{dv}[J]$, so we let σ' be the substitution obtained by composing the substitutions $\{^{\widetilde{y}}/_{\widetilde{z}}\}$ for all x in $\mathsf{dv}[J]$.

Let P'_1 be the process Q_1 where $[\![U\sigma']\!]^\circ$ is substituted for $[\![U\sigma']\!]$. We apply Lemma 16 to prove that $Q_1 \succeq P'_1$. To this end, we use the sets of names $X \cup L$ and F, the join-calculus process $P = U\sigma'$, the sjoin-calculus definition $D' = D_A \wedge [\![D]\!]$, and the sjoin-calculus process $Q = P_A \mid [\![T]\!] \mid [\![M'']\!]$. We rely on the invariants (5), (3), and (4) to satisfy the requirements on names in $\mathcal{B}^-$, $(\mathcal{B} \setminus (X \cup L))^\circ$, and $(\mathcal{B} \setminus (X \cup L))^+$ that appear in the statement of Lemma 16.

In P_2, after structural rearrangements in D_A and P_A, we can substitute $x\langle\widetilde{y}\rangle$ for every message $x^\circ\langle\widetilde{y}^+\rangle$ in M'_A (up to expansion) by applying Lemma 15(1), then move the message $x\langle\widetilde{y}\rangle$ from P_A to M (up to structural equivalence). Let P'_2 be the process obtained by composing these relations. We have $P_2 \succeq P'_2$, and there is now a reduction step $P'_2 \rightarrow Q_2$ that uses the rule $J \triangleright U$ to substitute the process $U\sigma'$ for $J\sigma'$.

We easily check that $P'_1\ \mathcal{R}\ P'_2$ and, by composing the relations above, we obtain $Q_1 \succeq P'_1\ \mathcal{R}\ P'_2 \leftarrow\preceq P_2$, hence $Q_1 \succeq\mathcal{R}\preceq\leftarrow^* P_2$.

- A reduction $P_2 \rightarrow Q_2$ uses the join-pattern of some rule $J \triangleright U$ in D to consume messages in M. As described above for Q_1, we can apply Lemma 16 and obtain $P_1 \rightarrow\succeq\mathcal{R}\equiv Q_2$.

Reductions in $[\![T]\!]$ and T: These reductions are comparisons: in P_2, they replace processes of the form $\mathsf{if}\ x = y\ \mathsf{then}\ U\ \mathsf{else}\ U'$ with either U or U', while in P_1 they replace translated processes of the form $\mathsf{if}\ x^+ = y^+\ \mathsf{then}\ [\![U]\!]\ \mathsf{else}\ [\![U']\!]$ with either $[\![U]\!]$ or $[\![U']\!]$. Since x and y have the same type, by assumption 3 on names we obtain that $x^+ = y^+$ if and only if $x = y$, hence that the same branch is chosen in P_1 and P_2. As in the previous case, Lemma 16 enables us to substitute $[\![U]\!]^\circ$ for $[\![U]\!]$ (or $[\![U']\!]^\circ$ for $[\![U']\!]$), and the two resulting processes are still related by $\mathcal{R}$.

Reductions using $D_{\Sigma(X,F)}$: The definition $D_{\Sigma(X,F)}$ binds the names $y \in F$, $x^\circ \in X^\circ$, and $t_\sigma, t'_\sigma \in \mathcal{T}$. Invariant 2 requires that the names in F appear only in D, M, T, and $D_{\Sigma(X,F)}$, and invariant 8 forbids messages sent on names in F in M, so P_2 has no reduction that consumes messages sent on names in F. Invariant 6 requires that the names in $\mathcal{T}$ occur only in $D_{\Sigma(X,F)}$, so P_2 has no reduction that involves the association tables of the firewall.

Let $P_2 \rightarrow Q_2$ be a reduction that consumes a message $x^\circ\langle\widetilde{v}\rangle$ in P_A, for some $x \in X$. In P_1 and Q_2, we first use structural equivalence as in the cases above to lift any binder on names in $\widetilde{v}$ from P_A to D_A.

The reduction $P_2 \rightarrow Q_2$ is the first reduction of the traversal of the firewall described in Lemma 15(1), so $Q_2 \succeq P'_2$ for some P'_2 derived from P_2 by first performing the structural rearrangement described above, then substituting the message $x\langle\widetilde{y}\rangle$ for $x^\circ\langle\widetilde{y}^+\rangle$, and finally moving the message $x\langle\widetilde{y}\rangle$ from P_A to M.

The process P_1 can match these structural rearrangements, with no reduction, so we obtain $P_1 \equiv\mathcal{R}'\preceq Q_2$. $\square$

Using these lemmas, we establish Theorem 3.

Proof of Theorem 3: For a given process P, we choose sets Σ, $\mathcal{T}$, $\mathcal{B}$, and $\mathcal{F}$, and functions $\cdot^+$, $\cdot^-$, and $\cdot^\circ$ that meet our assumptions on names of section B.5, and such that P has free names $F \subset \mathcal{F}$ and is strongly equivalent to $\mathsf{def}\ D\ \mathsf{in}\ Q$ where Q is a parallel composition of messages and of comparisons and where bound names that appear within D and Q are all in $\mathcal{B}$.

Taking $X = \emptyset$, $L = \mathsf{dv}[D]$, $F = \mathsf{fv}[P]$, D_A empty, and $P_A = \mathit{plug}\langle \mathit{emit}, \mathit{recv}\rangle$, the seven invariants of Lemma 17 directly follow from the assumptions on names, so we obtain the observational equivalence:

$$\begin{aligned} &\mathcal{E}_{L^\pm}\left[\mathsf{def}\ [\![D]\!]\ \mathsf{in}\ \mathit{plug}\langle \mathit{emit}, \mathit{recv}\rangle \mid [\![Q]\!]^\circ \mid W \mid \prod_{x\in L} R_x\right] \\ \approx\ \ &\mathcal{E}\left[\mathsf{def}\ D_{\Sigma(\emptyset,F)} \wedge D\ \mathsf{in}\ \mathit{plug}\langle \mathit{emit}, \mathit{recv}\rangle \mid Q \mid W\right]\end{aligned}$$

We write P_1 and P_2, respectively, for the two processes equated here, and we independently rearrange them to obtain the statement of the theorem.

- The processes W and $\mathit{plug}\langle \mathit{emit}, \mathit{recv}\rangle$ do not use the names x^+, x^-, or x° for $x \in L$. By rearranging the scopes of these names, and by the definitions of the auxiliary translation $[\![\ \cdot\ ]\!]^\circ$ and of the context $\mathcal{E}nv\,[\ \cdot\ ]$, we have the structural equivalence $P_1 \equiv \mathcal{E}nv\,[\,[\![P]\!]^\circ\,]$.

Applying Lemma 16 for $X = \emptyset$, D' empty, and $Q = plug\langle emit, recv\rangle$, we obtain the expansion relation $\mathcal{E}nv\,[\,[\![P]\!]\,] \succeq \mathcal{E}nv\,[\,[\![P]\!]^\circ\,]$, thus we have $P_1 \equiv\preceq \mathcal{E}nv\,[\,[\![P]\!]\,]$.

- On the other side, using the definition of $\mathcal{F}_F[P]$ we have:

$$\mathcal{F}_F[P] \quad \stackrel{\text{def}}{=} \quad \mathsf{def}\ D_{\Sigma(\emptyset,F)}\ \mathsf{in}\ P\ \equiv\ \mathsf{def}\ D_{\Sigma(\emptyset,F)} \wedge D\ \mathsf{in}\ Q$$

  (where the structural equivalence rearranges the scopes of names defined in D, since these names do not appear in $D_{\Sigma(\emptyset,F)}$). By application of the context $\mathcal{E}nv\,[\,\cdot\,]$, we obtain $\mathcal{E}nv\,[\,\mathcal{F}_F[P]\,] \equiv P_2$.

In combination, these relations yield $\mathcal{E}nv\,[\,[\![P]\!]\,] \approx \mathcal{E}nv\,[\,\mathcal{F}_F[P]\,]$. □

The next lemma provides an inverse firewall $\mathcal{G}_F[\,\cdot\,]$ for the firewall $\mathcal{F}_F[\,\cdot\,]$, up to expansion, for a given choice of names F.

Lemma 18 *For a finite set of names $F \subset \mathcal{F}$, we define the context:*

$$\mathcal{G}_F[\,\cdot\,] \quad \stackrel{\textit{def}}{=} \quad \mathsf{def} \bigwedge_{x \in F} \mathsf{keys}\ x^+, x^- \wedge D_{\Sigma(F,\emptyset)}\ \mathsf{in} \prod_{x \in F} R_x \mid [\,\cdot\,]$$

For every join-calculus process P such that $\mathsf{fv}[P] \subseteq F$, we have the expansion relations:

$$\mathcal{E}\,[\,\mathcal{G}_F\,[\mathcal{F}_F[P]] \mid W\,] \quad \succeq \quad P \tag{1}$$

$$\mathcal{E}nv\,[\,\mathcal{G}_F\,[\mathcal{F}_F[P]]\,] \quad \succeq \quad \mathcal{E}nv\,[\,P\,] \tag{2}$$

Proof: We refer to $\mathcal{F}_F[\,\cdot\,]$ and its derivatives as the "internal firewall" and to $\mathcal{G}_F[\,\cdot\,]$ and its derivatives as the "external firewall". For every free name x that appears in P, there are two names x in $\mathcal{E}\,[\,\mathcal{G}_F\,[\mathcal{F}_F[P]]\,]$: the original x of P that is now defined in $\mathcal{F}_F[\,\cdot\,]$, and a distinct, free name x that reappears in $\mathcal{G}_F\,[\,\cdot\,]$. To avoid name clashes, we α-convert every internal name x to $\overline{x}$.

We adapt and extend our notations. As in the previous lemmas, we assume a fixed, finite set of join-calculus types Σ and make explicit our assumptions on names. We distinguish four disjoint sets of names $\mathcal{B}$, $\overline{\mathcal{B}}$, $\mathcal{F}$, and $\overline{\mathcal{F}}$ such that, for every type $\sigma \in \Sigma$, there is an infinite number of names in each set, and a bijective function $\overline{\cdot}$ from names in $\mathcal{B}$ and $\mathcal{F}$ to names in $\overline{\mathcal{B}}$ and $\overline{\mathcal{F}}$, respectively. We also distinguish two disjoint sets of table names $\mathcal{T} = \bigcup_{\sigma \in \Sigma}\{t_\sigma, t'_\sigma\}$ and $\overline{\mathcal{T}} = \bigcup_{\sigma \in \Sigma}\{\overline{t}_\sigma, \overline{t}'_\sigma\}$. We use injective functions $\cdot^+$, $\cdot^-$, and $\cdot^\circ$ with domain $\mathcal{B} \uplus \mathcal{F}$ that range over disjoint sets of names. We assume that $(\mathsf{fv}[E_x[\,\cdot\,]] \cup \mathsf{fv}[R_x]) \cap ((X \cup F)^+ \cup (X \cup F)^-) \subseteq \{x^+, x^-\}$ for all $x \in X \cup F$. We also assume an infinite number of names of every sjoin-calculus type outside these sets and their images by these functions.

We overline parts of the internal firewall that wraps P. We still use the notation $D_{\Sigma(F,X)}$ for the tables and filtering definitions of the external firewall (where the two sets X and F have been swapped with respect to the previous proofs). We also use a similar notation for the tables and filtering definitions of the internal firewall: for all disjoint sets of names X and F, we write $\overline{D}_\Sigma(X \cup F)$ instead of $D_\Sigma(X \cup F)$ for the definition of association tables that contain, for every type $\sigma \in \Sigma$, the pairs $(\overline{x}, x^+)$ for all $x \in X \cup F$ of type σ, and that are accessed through the names $\overline{t}_\sigma$ and $\overline{t}'_\sigma$ instead of t_σ and t'_σ. Gathering all the definitions of the internal firewall, we let

$$\overline{D}_{\Sigma(X,F)} \quad \stackrel{\text{def}}{=} \quad \overline{D}_\Sigma(X \cup F) \wedge \bigwedge_{x \in X} D_{x^\circ \triangleright \overline{x}} \wedge \bigwedge_{x \in F} D_{\overline{x} \triangleright E_x}$$

Hence, we have $\mathsf{dv}[D_{\Sigma(F,X)}] = \mathcal{T} \cup X \cup F^\circ$ and $\mathsf{dv}[\overline{D}_{\Sigma(X,F)}] = \overline{\mathcal{T}} \cup \overline{F} \cup X^\circ$.

We prove an expansion relation more general than the second statement of the lemma, which concerns a process with two opposite firewalls and a process with no firewalls. We let $\mathcal{R}$ be the relation such that:

$$\begin{array}{ll} \mathcal{E}_{(X \cup F)^\pm} \left[\begin{array}{llll} & D_{\Sigma(F,X)} & & \prod_{x \in X} R_x \mid P' \\ \mathsf{def} \ \wedge & \overline{D}_{\Sigma(X,F)} & \mathsf{in} \ \mid & \prod_{x \in F} R_x \mid \overline{P} \\ \wedge & D' \wedge \overline{D} & \mid & W \end{array} \right] \\ \mathcal{R} \quad \mathcal{E}\left[\mathsf{def}\ D' \wedge D \ \mathsf{in}\ P' \mid P \mid W \right] \end{array}$$

provided that the following invariants hold.

1. $\mathsf{def}\ D\ \mathsf{in}\ P$ is a join-calculus process.
2. $X \subseteq \mathsf{dv}[D] \subset \mathcal{B}$ and $\mathsf{fv}[\mathsf{def}\ D\ \mathsf{in}\ P] \subseteq F \subset \mathcal{F}$.
3. $\overline{D}$ and $\overline{P}$ are obtained from D and P by overlining every name.
4. $\mathsf{dv}[D'] \cap \mathcal{B} = \emptyset$, and names in $\mathsf{fv}[D'] \cup \mathsf{fv}[P']$ of types in Σ are all in $X \cup \mathcal{F}$.
5. Names in $\overline{\mathcal{B}} \cup \overline{\mathcal{F}}$ occur only in $\overline{D}_{\Sigma(X,F)}$, $\overline{D}$, and $\overline{P}$.
6. Names in $(\mathcal{B} \cup \mathcal{F})^+$, $(\mathcal{B} \cup \mathcal{F})^-$, $(\mathcal{B} \cup \mathcal{F})^\circ$, $\mathcal{T}$, $\overline{\mathcal{T}}$ do not occur in D', P', R_x (except for x^+, x^-, x° in R_x).

We show that $\mathcal{R} \subseteq \succeq$. Let $P_1\ \mathcal{R}\ P_2$. Before developing a bisimulation argument, we study the reductions of P_1 that describe the filtering of messages from P' to $\overline{P}$, or vice versa. We show that these reductions are all included in the expansion relation.

(i) Incoming messages: Let $x\langle\widetilde{y}\rangle$ be a message in an evaluation context in P' such that $x \in X$ and x is not bound in P. We choose D'' and P'' such that $P' \equiv \mathsf{def}\ D''\ \mathsf{in}\ x\langle\widetilde{y}\rangle \mid P''$, that $\mathsf{dv}[D''] \cap \mathsf{dv}[D'] = \emptyset$, and that the invariants 4–6 are preserved when we substitute $D' \wedge D''$ for D' and $x\langle\widetilde{y}\rangle \mid P''$ for P'. We use structural equivalence to perform this substitution in both P_1 and P_2. In P_1, we have the following behavior:

1. The message $x\langle\widetilde{y}\rangle$ is received and filtered by definitions in $D_{\Sigma(F,X)}$. Let F' be the set $F \cup (\{\widetilde{y}\} \setminus (X \cup F))$. Since $x \in X$, we can use the expansion relation of Lemma 15(2) to replace $x\langle\widetilde{y}\rangle$ with $E_x[\widetilde{y}^+]$, $D_{\Sigma(F,X)}$ with $D_{\Sigma(F',X)}$, and $\prod_{x\in F} R_x$ with $\prod_{x\in F'} R_x$.
2. The new emission protocol $E_x[\widetilde{y}^+]$ is running in parallel with the decryption protocol R_x. Since x^- appears only in R_x, we can apply Lemma 5 to the correctness condition 4 on the protocol. The resulting expansion relation replaces the process $E_x[\widetilde{y}^+]$ with the message $x^\circ\langle\widetilde{y}^+\rangle$.
3. The message $x^\circ\langle\widetilde{y}^+\rangle$ is received and filtered by definitions in $\overline{D}_{\Sigma(X,F)}$. Since $x \in X$, these operations are captured by the expansion relation of Lemma 15(1), which replaces $x^\circ\langle\widetilde{y}^+\rangle$ with $\overline{x\langle\widetilde{y}\rangle}$ and yields the definitions $\overline{D}_{\Sigma(X,F')}$ with the same F' as above.

In summary, we compose three expansion relations for every message $x\langle\widetilde{y}\rangle$ with $x \in X$ that appears in an evaluation context in P'; by transitivity of expansion we obtain $P_1 \succeq P_1'$ where P_1' is obtained from P_1 by substituting $D' \wedge D''$ for D', P'' for P', and $\overline{x\langle\widetilde{y}\rangle \mid P}$ for $\overline{P}$. In P_2, we use structural equivalence to move the message $x\langle\widetilde{y}\rangle$ from P' to P so that the two resulting processes are still related ($P_1 \succeq P_1' \mathrel{\mathcal{R}} P_2' \equiv P_2$).

(o) Outgoing messages: Let $x\langle\widetilde{y}\rangle$ be a message in an evaluation context in P such that $x \in F$ and x is not bound in P. In a similar manner, we choose D'' and P'' such that $P \equiv \mathsf{def}\ D''\ \mathsf{in}\ x\langle\widetilde{y}\rangle \mid P''$, $\mathsf{dv}[D''] \subseteq (\mathcal{B} \setminus \mathsf{dv}[D])$, and invariants 1 and 2 are preserved, then we use structural equivalence to lift D'' to D in both P_1 and P_2.

In P_1, we have a series of three expansions, as the message is received and filtered by $\overline{D}_{\Sigma(X,F)}$ (using Lemma 15(2)), as it is transmitted using a run of the communication protocol (using Lemma 5 and the correctness condition 4 on the protocol), and as it is received and filtered by $D_{\Sigma(F,X)}$ (using Lemma 15(1)).

Much as in the case of incoming messages, we build a pair of processes P_1' and P_2' in which the message $x\langle\widetilde{y}\rangle$ has been transferred to P' and the set X has been extended with $\{\widetilde{y}\}\setminus(X\cup F)$, such that $P_1 \succeq P_1' \mathrel{\mathcal{R}} P_2' \equiv P_2$.

We apply Lemma 3 to establish that $\mathcal{R} \subseteq \succeq$. We let $\mathcal{N}'$ gather all names in $\{x \text{ of type } \sigma \mid \sigma \in \Sigma\} \setminus \mathcal{F}$, $\mathcal{B}^+$, $\mathcal{B}^-$, $\mathcal{B}^\circ$, $\mathcal{F}^+$, $\mathcal{F}^-$, $\mathcal{F}^\circ$, $\mathcal{T}$, $\overline{\mathcal{T}}$, and $\{emit, recv\}$. We easily check that $\mathcal{R}$ meets properties 5 and 2 of Lemma 3.

We compare the barbs of processes related by $\mathcal{R}$. By correctness conditions 2 and 3 on the protocol, the receiving processes R_x may emit messages only on *recv*. By definition, W emits messages only on *emit*. By invariants 2 and 3, we have $\mathsf{fv}[\overline{P}] \subseteq \overline{F} \cup \overline{X} \subseteq \mathsf{dv}[\overline{D}_{\Sigma(X,F)}] \cup \mathsf{dv}[\overline{D}]$, so $\overline{P}$ has no message on names that are free in P_1. By invariant 6, definitions present in P_1 but not in P_2 cannot bind names free in P', so P' provides the same strong barbs to P_1 and P_2. We still have to prove that if $P_2 \downarrow_x$ because $P \downarrow_x$ for some $x \in F \setminus \mathsf{dv}[D']$, then $P_1 \Downarrow_x$. To this end, we use the series of expansions (o) described above to obtain $P_1 \succeq P_1'$ with $P_1' \downarrow_x$, hence $P_1 \Downarrow_x$.

To establish the bisimulation properties, we consider all reduction steps of the form $P_1 \rightarrow Q_1$ and $P_2 \rightarrow Q_2$. The proof is by cases, relying on the following classification of reductions: reductions that involve only D', P', W, $\mathcal{E}[\,\cdot\,]$, reductions that use a join-pattern of D or $\overline{D}$, reductions internal to P or $\overline{P}$, reductions that use a join-pattern of D', and, in P_1 only, reductions that involve a process R_x, reductions that involve a definition of $D_{\Sigma(F,X)}$, and reductions that involve a definition of $\overline{D}_{\Sigma(X,F)}$.

Reductions that involve only D', P', W, and $\mathcal{E}[\,]$: Such reductions are in direct correspondence. Using structural rearrangement, we have $Q_1 \equiv P_1' \mathrel{\mathcal{R}} P_2' \equiv Q_2$ for some processes P_1' and P_2' obtained from P_1 and P_2 for some new D' and P'.

Reductions using a join-pattern of $\overline{D}$ (in P_1) or D (in P_2): If $P_1 \rightarrow Q_1$ is such a reduction, by invariant 5 it consumes messages $\overline{M}$ in $\overline{P}$ only, so there is a corresponding reduction $P_2 \rightarrow Q_2$ that consumes messages M in P. We obtain $Q_1 \equiv\mathcal{R}\equiv Q_2$.

Conversely, if $P_2 \rightarrow Q_2$, this reduction may consume messages from both P and P', so P_1 may not be able to match it immediately. Let us assume that $P' \equiv \mathsf{def}\ D''\ \mathsf{in}\ M' \mid P''$ where M' is a parallel composition of the messages of P' consumed in that reduction, and where $\mathsf{dv}[D'']$ meets invariant 4–6 and is disjoint from $\mathsf{dv}[D']$. Starting from P_1, for every message in M' we apply the expansions (i) described above to transform the message from P' into an overlined message in parallel with $\overline{P}$. Once all messages have been transferred and the set F has been extended with $\mathsf{fv}[M'] \setminus (X \cup F)$, there is a reduction $P_1 \rightarrow P_1'$ in direct correspondence with the reduction $P_2 \rightarrow Q_2$. We obtain $P_1 \succeq\rightarrow P_1' \mathrel{\mathcal{R}} P_2' \equiv Q_2$, hence $P_1 \rightarrow^*\succeq P_1' \mathrel{\mathcal{R}} P_2' \equiv Q_2$

Reductions internal to $\overline{P}$ or P: Such reductions are in direct correspondence by overlining, and do not affect the invariants.

Reductions using a join-pattern in D': If $P_1 \rightarrow Q_1$ is such a reduction, all messages being consumed must appear in P', and we immediately have a matching reduction $P_2 \rightarrow Q_2$.

Conversely, the reduction $P_2 \rightarrow Q_2$ may consume messages from both P and P'; much as in the case of a reduction using D, in P_1 messages from $\overline{P}$ are first transferred to P' using the expansions (o) described above, then the resulting process can perform the same reduction (in different context). We obtain $P_1 \succeq \rightarrow \mathcal{R} \equiv Q_2$, hence $P_1 \rightarrow^* \succeq \mathcal{R} \equiv Q_2$.

Reductions that involve the receiving processes: Such reductions occur only in P_1 and involve a receiving process in its initial state R_x for some $x \in X \cup F$, so they are receptions of some message $\mathit{emit}\langle m \rangle$ (or they are internal to R_x, but this case is subsumed by receptions). Moreover, this message $\mathit{emit}\langle m \rangle$ must be in $P' \mid W$, thus m does not contain the keys x^+ or x^- (by invariant 6). Correctness condition 5 on the protocol applies; we reformulate its statement by Lemma 5 for the current context, and obtain that, for all reductions $P_1 \rightarrow Q_1$, we also have $Q_1 \succeq P_1$. Such reductions do not require any reduction in P_2.

Reductions that involve $D_{\Sigma(F,X)}$: By invariant 6, the names F° may appear only in $D_{\Sigma(F,X)}$. By construction, the names in X may not appear in $\overline{P}$. We still have to consider all reductions that consume messages $x\langle \widetilde{y} \rangle$ in P', in the case $x \in X$. These are the initial reductions described by the expansions (i), so we immediately obtain $Q_1 \succeq \mathcal{R} \equiv P_2$.

Reductions that involve $\overline{D}_{\Sigma(X,F)}$: By invariant 6, the names X° may appear only in $\overline{D}_{\Sigma(X,F)}$. Moreover, the names in $\overline{F}$ may occur only in $\overline{D}$ and $\overline{P}$. We still have to consider all reductions that consume messages $\overline{x\langle \widetilde{y} \rangle}$ in $\overline{P}$, in the case $x \in F$. These are the initial reductions described by the expansions (o), so we immediately obtain $Q_1 \succeq \mathcal{R} \equiv P_2$.

For any join-calculus process P such that $\mathsf{fv}[P] \subseteq F$, we obtain the two statements of the lemma from the inclusion $\mathcal{R} \subseteq \succeq$. We take D, D', and X empty, and either $P' = \mathit{plug}\langle \mathit{emit}, \mathit{recv} \rangle$ or $P' = \mathsf{0}$. We have

$$
\begin{array}{ll}
 & \mathcal{E}\left[\, \mathcal{G}_F\left[\mathcal{F}_F[P]\right] \mid P' \mid W \,\right] \\
\equiv & \mathcal{E}_{F^\pm}\left[\mathsf{def}\ D_{\Sigma(F,\emptyset)}\ \mathsf{in} \prod_{x \in F} R_x \mid (\mathsf{def}\ \overline{D}_{\Sigma(\emptyset,F)}\ \mathsf{in}\ \overline{P}) \mid P' \mid W \right] \\
\equiv & \mathcal{E}_{F^\pm}\left[\mathsf{def}\ D_{\Sigma(F,\emptyset)} \wedge \overline{D}_{\Sigma(\emptyset,F)}\ \mathsf{in} \prod_{x \in F} R_x \mid \overline{P} \mid P' \mid W \right] \\
\mathcal{R} & \mathcal{E}\left[\, P \mid P' \mid W \,\right]
\end{array}
$$

where the structural equivalence first performs α-conversion to overline every name that appears in P or that is defined in $\mathcal{F}_F[\,\cdot\,]$, then flattens the definitions.

In the case where P' is $plug\langle emit, recv\rangle$, we obtain statement (2) by definition of $\mathcal{E}nv\,[\,\cdot\,]$ and structural equivalence. In the case where P' is $\mathsf{0}$, we obtain $\mathcal{E}\,[\,\mathcal{G}_F\,[\mathcal{F}_F[P]]\mid W\,] \succeq \mathcal{E}\,[\,P \mid W\,]$. Since P is a join-calculus process, we also have $\mathcal{E}\,[\,P \mid W\,] \equiv P \mid \mathcal{E}\,[\,W\,]$. Finally, $\mathcal{E}\,[\,W\,] \succeq \mathsf{0}$ because this process has no free variable. Composing these relations, we obtain statement (1). □

Lemma 19 states that both translations are fully abstract for all transitive relations coarser than observational equivalence and closed by application of evaluation contexts. This lemma implies Theorems 2 and 4.

Lemma 19 *Let $\mathcal{R}$ be a transitive relation on sjoin-calculus processes that is closed by application of evaluation contexts, and such that $\approx\,\subseteq \mathcal{R}$. For all join-calculus processes P and Q, for all finite sets F of names with $\mathsf{fv}[P] \cup \mathsf{fv}[Q] \subseteq F$,*

1. *$P \mathrel{\mathcal{R}} Q$ if and only if $\mathcal{E}nv\,[\,\mathcal{F}_F[P]\,] \mathrel{\mathcal{R}} \mathcal{E}nv\,[\,\mathcal{F}_F[Q]\,]$,*
2. *$P \mathrel{\mathcal{R}} Q$ if and only if $\mathcal{E}nv\,[\,[\![P]\!]\,] \mathrel{\mathcal{R}} \mathcal{E}nv\,[\,[\![Q]\!]\,]$.*

Proof: We first prove statement (1). Since $\mathcal{E}nv\,[\,\mathcal{F}_F[\,\cdot\,]\,]$ is an evaluation context, if $P \mathrel{\mathcal{R}} Q$, then $\mathcal{E}nv\,[\,\mathcal{F}_F[P]\,] \mathrel{\mathcal{R}} \mathcal{E}nv\,[\,\mathcal{F}_F[Q]\,]$. In order to establish the converse, we use the evaluation context

$$D_F[\,\cdot\,] \quad \stackrel{\text{def}}{=} \quad \mathsf{def} \quad \begin{array}{ll} & \bigwedge_{x \in F} \mathsf{keys}\ x^+, x^- \\ \wedge & plug\langle emit, recv\rangle \triangleright (\mathsf{def}\ D_{\Sigma(F,\emptyset)}\ \mathsf{in}\ \prod_{x \in F} R_x) \end{array} \quad \mathsf{in}\ [\,\cdot\,]$$

For every join-calculus process T such that $\mathsf{fv}[T] \subseteq F$, we have the relations

$$\begin{array}{rcl} D_F[\mathcal{E}nv\,[\,\mathcal{F}_F[T]\,]] & \equiv & D_F[\mathcal{E}\,[\,plug\langle emit, recv\rangle\,] \mid \mathcal{F}_F[T] \mid W] \\ & \rightarrow\sim & \mathcal{E}_F\left[(\mathsf{def}\ D_{\Sigma(F,\emptyset)}\ \mathsf{in}\ \displaystyle\prod_{x \in F} R_x) \mid \mathcal{F}_F[T] \mid W\right] \\ & \equiv & \mathcal{E}\,[\,\mathcal{G}_F[\mathcal{F}_F[T] \mid W]\,] \end{array}$$

By applying the special case of Lemma 3, it is straightforward to prove the expansion relation $D_F[\mathcal{E}nv\,[\,\mathcal{F}_F[T]\,]] \succeq \mathcal{E}\,[\,\mathcal{G}_F[\mathcal{F}_F[T]] \mid W\,]$. Besides, by Lemma 18, we have another expansion relation $\mathcal{E}\,[\,\mathcal{G}_F[\mathcal{F}_F[T]] \mid W\,] \succeq T$.

If $\mathcal{E}nv\,[\,\mathcal{F}_F[P]\,] \;\mathcal{R}\; \mathcal{E}nv\,[\,\mathcal{F}_F[Q]\,]$ then, since $\mathcal{R}$ is closed by application of evaluation contexts, we have $D_F[\mathcal{E}nv\,[\,\mathcal{F}_F[P]\,]] \mathrel{\mathcal{R}} D_F[\mathcal{E}nv\,[\,\mathcal{F}_F[Q]\,]]$. By composing $\mathcal{R}$ and the expansions above with P and Q substituted for T, we obtain $P \preceq\mathcal{R}\succeq Q$, so $P \mathrel{\mathcal{R}\mathcal{R}\mathcal{R}} Q$ and $P \mathrel{\mathcal{R}} Q$.

Statement (2) of the lemma is a corollary of statement (1) in combination with Theorem 3. □

B.6 Compositionality of the filtering translation

Finally, we prove Theorem 6 as a corollary of the results proved above. Theorem 5 immediately follows from Theorem 6.

Proof of Theorem 6: We make use of a "+" operator on definitions in the join-calculus: to every definition $D = \bigwedge_k J_k \triangleright P_k$ and message $\rho\langle\widetilde{y}\rangle$ such that the names $\rho, \widetilde{y}$ do not appear in any J_k, we associate the definition $D + \rho\langle\widetilde{y}\rangle \stackrel{\text{def}}{=} \bigwedge_k (J_k \mid \rho\langle\widetilde{y}\rangle) \triangleright (P_k \mid \rho\langle\widetilde{y}\rangle)$. Provided that $\rho \notin \mathsf{fv}[D] \cup \mathsf{fv}[P]$, we have the strong equivalence $\mathsf{def}\ D\ \mathsf{in}\ P \sim \mathsf{def}\ D + \rho\langle\widetilde{y}\rangle\ \mathsf{in}\ P \mid \rho\langle\widetilde{y}\rangle$.

We adopt the same hypotheses and naming conventions as in section B.5. In the statement of Theorem 6, let $P = \mathsf{def}\ \bigwedge_{i=1}^n D_i\ \mathsf{in}\ \prod_{i=1}^n Q_i$ in the join-calculus, and let S be a finite set of names such that $\mathsf{fv}[P] \subseteq S$. For $i = 1 \dots n$, we choose tuples $\widetilde{x}_i$ and $\widetilde{x}^i$ that contain the names in the sets X_i and X^i, respectively, and also names $\rho_i, s_i, z_i \in \mathcal{B}$ that are not free in D_i, Q_i, or S. We let

$$\begin{aligned}
D_r &\stackrel{\text{def}}{=} \prod_{i=1}^n z_i\langle\widetilde{x}_i, s_i\rangle \triangleright \prod_{i=1}^n s_i\langle\widetilde{x}^i\rangle \\
D'_i &\stackrel{\text{def}}{=} (D_i + \rho_i\langle\widetilde{x}^i\rangle) \wedge (s_i\langle\widetilde{x}^i\rangle \triangleright \rho_i\langle\widetilde{x}^i\rangle \mid Q_i) \\
P'_i &\stackrel{\text{def}}{=} \mathsf{def}\ D'_i\ \mathsf{in}\ z_i\langle\widetilde{x}_i, s_i\rangle \\
P' &\stackrel{\text{def}}{=} \mathsf{def}\ D_r\ \mathsf{in}\ \prod_{i=1}^n P'_i
\end{aligned}$$

Thus, P' differs from P because the n components P'_i do not initially share names and instead exchange names through the definition D_r, before behaving like the original components of P.

Next, we establish the equivalence $P \approx P'$. In P', the reduction that consumes the messages $z_i\langle\widetilde{x}_i, s_i\rangle$ and triggers the messages $s_i\langle\widetilde{x}^i\rangle$ is deterministic, then the n reductions that consume the messages $s_i\langle\widetilde{x}^i\rangle$ and trigger the processes $\rho_i\langle\widetilde{x}^i\rangle \mid Q_i$ are also deterministic. Finally, the names z_i and s_i do not appear any more except in their definitions, so they can be discarded up to strong equivalence, and we obtain P by composing n strong equivalences $\mathsf{def}\ D_i + \rho_i\langle\widetilde{x}^i\rangle\ \mathsf{in}\ Q_i \mid \rho_i\langle\widetilde{x}^i\rangle \sim \mathsf{def}\ D_i\ \mathsf{in}\ Q_i$.

We transfer this equivalence to the sjoin-calculus. We have the following series of relations:

$$\begin{aligned}
& \mathcal{E}nv\,[\,\mathcal{F}_S[P]\,] \\
\approx\ & \mathcal{E}nv\,[\,\mathcal{F}_S[P']\,] && (1) \\
\approx\ & \mathcal{E}nv\,[\,[\![P']\!]\,] && (2) \\
\equiv\ & \mathcal{E}nv_{\widetilde{z}^\pm}\left[\mathsf{def}\ [\![D_r]\!]\ \mathsf{in}\ \prod_{i=1}^n (R_{z_i} \mid [\![P'_i]\!])\right] && (3)
\end{aligned}$$

$$\approx \quad \mathcal{E}nv_{\widetilde{z}^\pm}\left[\mathsf{def}\ [\![D_r]\!]\ \mathsf{in}\ \prod_{i=1}^{n}\Big(R_{z_i} \mid \mathcal{F}_{S\cup\{z_i\}}[P'_i]\Big)\right] \tag{4}$$

$$\equiv \quad \mathcal{E}nv_{\widetilde{z}^\pm}\left[\mathsf{def}\ [\![D_r]\!]\ \mathsf{in}\ \prod_{i=1}^{n}\Big(R_{z_i} \mid \mathcal{F}^{\emptyset}_{S\cup\{z_i\}}[D'_i,\ z_i\langle\widetilde{x}_i, s_i\rangle]\Big)\right] \tag{5}$$

$$\succeq \quad \mathcal{E}nv_{X^\pm,\widetilde{s}^\pm,\widetilde{z}^\pm}\left[\mathsf{def}\ [\![D_r]\!]\ \mathsf{in}\ \prod_{i=1}^{n}\Big(R_{z_i} \mid E_{z_i}[\widetilde{x}_i^+, s_i^+] \mid \mathcal{F}^{X_i\cup\{s_i\}}_{S\cup\{z_i\}}[D'_i,\ \mathsf{0}]\Big)\right] \tag{6}$$

$$\succeq \quad \mathcal{E}nv_{X^\pm,\widetilde{s}^\pm,\widetilde{z}^\pm}\left[\mathsf{def}\ [\![D_r]\!]\ \mathsf{in}\ \prod_{i=1}^{n}\Big(R_{z_i} \mid z_i^\circ\langle\widetilde{x}_i^+, s_i^+\rangle \mid \mathcal{F}^{X_i\cup\{s_i\}}_{S\cup\{z_i\}}[D'_i,\ \mathsf{0}]\Big)\right] \tag{7}$$

$$\succeq \quad \mathcal{E}nv_{X^\pm,\widetilde{s}^\pm,\widetilde{z}^\pm}\left[\mathsf{def}\ [\![D_r]\!]\ \mathsf{in}\ \prod_{i=1}^{n}\Big(R_{z_i} \mid E_{s_i}[\widetilde{x}^{i+}] \mid \mathcal{F}^{X_i\cup\{s_i\}}_{S\cup\{z_i\}}[D'_i,\ \mathsf{0}]\Big)\right] \tag{8}$$

$$\succeq \quad \mathcal{E}nv_{X^\pm,\widetilde{s}^\pm}\left[\prod_{i=1}^{n}\Big(E_{s_i}[\widetilde{x}^{i+}] \mid \mathcal{F}^{X_i\cup\{s_i\}}_{S}[D'_i,\ \mathsf{0}]\Big)\right] \tag{9}$$

$$\succeq \quad \mathcal{E}nv_{X^\pm,\widetilde{s}^\pm}\left[\prod_{i=1}^{n}\mathcal{F}^{X_i\cup\{s_i\}}_{S}\Big[D'_i,\ s_i^\circ\langle\widetilde{x}^{i+}\rangle\Big]\right] \tag{10}$$

$$\succeq \quad \mathcal{E}nv_{X^\pm,\widetilde{s}^\pm}\left[\prod_{i=1}^{n}\mathcal{F}^{X_i\cup\{s_i\}}_{S\cup X^i}\Big[D'_i,\ s_i\langle\widetilde{x}^{i}\rangle\Big]\right] \tag{11}$$

$$\approx \quad \mathcal{E}nv_{X^\pm,\widetilde{s}^\pm}\left[\prod_{i=1}^{n}\mathcal{F}^{X_i\cup\{s_i\}}_{S\cup X^i}[D_i,\ Q_i]\right] \tag{12}$$

$$\succeq \quad \mathcal{E}nv_{X^\pm}\left[\prod_{i=1}^{n}\mathcal{F}^{X_i}_{S\cup X^i}[D_i,\ Q_i]\right] \tag{13}$$

The equivalence (1) is the join-calculus equivalence $P \approx P'$ in the evaluation context $\mathcal{E}nv\,[\,\mathcal{F}_S[\,\cdot\,]\,]$. The equivalence (2) is Theorem 3 applied to P'. The structural rearrangement (3) follows from the definitions of P' and of the compositional translation $[\![\,\cdot\,]\!]$. The equivalence (4) is obtained by composing n applications of Theorem 2 for each process P'_i in combination with Lemma 5. The structural rearrangement (5) follows from the definitions of the firewall contexts $\mathcal{F}[\,\cdot\,]$ and $\mathcal{F}[\,\cdot,\ \cdot\,]$. The expansion (6) accounts for the traversals of the ith firewall for every message $s_i^\circ\langle\widetilde{x}^{i+}\rangle$; it uses n times Lemma 15(2). The expansion (7) is composed of n runs of the secure communication protocol on $z_1, \ldots, z_n$, as described by correctness condition 4 on the protocol, in combination with Lemma 5. The expansion (8) stands for the deterministic reduction that uses the join-pattern of $[\![D_r]\!]$, consumes the process $\prod_{i=1}^{n} z_i\langle\widetilde{x}_i, s_i\rangle$, and produces the process $[\![\prod_{i=1}^{n} s_i\langle\widetilde{x}^i\rangle]\!] = \prod_{i=1}^{n}[\![s_i\langle\widetilde{x}^i\rangle]\!] = \prod_{i=1}^{n} E_{s_i}[\widetilde{x}^{i+}]$. The expansion (9) garbage-collects the names z_i, in three stages: (a) since z_i occurs only in the definition $D_{z_i \triangleright E_{z_i}}$ in the ith firewall, we can discard this definition and the entry (z_i, z_i^+) from the single association table where it appears, up to strong equivalence; (b) since z_i^+ and z_i^- occur only in R_{z_i}, we can substitute $\mathsf{0}$ for R_{z_i} up to expansion, by correctness condition 5; (c) the names z_i° occur only in $[\![D_r]\!]$; the names z_i^+ and z_i^- occur only in their definitions $\mathsf{keys}\ z_i^+, z_i^-$ in $\mathcal{E}nv_{X^\pm,\widetilde{s}^\pm,\widetilde{z}^\pm}[\,\cdot\,]$; these definitions can be discarded up to strong equivalences. The expansion (10) is composed of n runs of the secure communication protocol on $s_1, \ldots, s_n$, as described by correctness condition 4 on the protocol, with Lemma 5. The expansion (11) accounts for the traversals of the firewalls for every message $s_i\langle\widetilde{x}^i\rangle$; it uses Lemma 15(1) n times. The equivalence (12) is the composition, for $i = 1, \ldots, n$, of a deterministic reduction that consumes the message $s_i\langle\widetilde{x}^i\rangle$ and of a strong equivalence that discards $+\rho_i\langle\widetilde{x}^i\rangle$. Finally, the expansion (13) garbage-collects the names s_i, much as in step (9). $\square$

References

[1] Martín Abadi. Protection in programming-language translations. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 868–883. Springer-Verlag, July 1998.

[2] Martín Abadi, Andrew Birrell, Raymie Stata, and Edward Wobber. Secure web tunneling. *Computer Networks and ISDN Systems*, 30(1–7):531–539, April 1998. Proceedings of the 7th International World Wide Web Conference.

[3] Martín Abadi, Cédric Fournet, and Georges Gonthier. Secure implementation of channel abstractions. In *Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science*, pages 105–116, June 1998.

[4] Martín Abadi, Cédric Fournet, and Georges Gonthier. Secure communications processing for distributed languages. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 74–88, May 1999.

[5] Martín Abadi, Cédric Fournet, and Georges Gonthier. A top-down look at a secure message. In *Proceedings of the 19th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 122–141. Springer-Verlag, December 1999.

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

[7] Kenneth F. Alden and Edward P. Wobber. The AltaVista tunnel: Using the Internet to extend corporate networks. *Digital Technical Journal*, 9(2):5–15, October 1997. On the Web at `http://www.digital.com/info/DTJQ01/DTJQ01HM.HTM`.

[8] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[9] Andrew D. Birrell. Secure communication using remote procedure calls. *ACM Transactions on Computer Systems*, 3(1):1–14, February 1985.

[10] Michele Boreale, Cédric Fournet, and Cosimo Laneve. Bisimulations in the join-calculus. In *IFIP Working Conference on Programming Concepts and Methods (PROCOMET'98)*, pages 68–86. Chapman and Hall, June 1998.

[11] Ed Brinksma, Arend Rensink, and Walter Vogler. Fair testing. In *Proceedings of CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 313–327. Springer-Verlag, 1995.

[12] Pau-Chen Cheng, Juan A. Garay, Amir Herzberg, and Hugo Krawczyk. Design and implementation of modular key management protocol and IP secure tunnel on AIX. In *Proceedings of the 5th USENIX UNIX Security Symposium*, June 1995.

[13] *10th IEEE Computer Security Foundations Workshop.* IEEE Computer Society, 1997. Proceedings.

[14] Mads Dam. Proving trust in systems of second-order processes. In *Proceedings of the 31th Hawaii International Conference on System Sciences*, volume VII, pages 255–264, 1998.

[15] Rocco De Nicola and Matthew C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[16] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[17] Cédric Fournet. *The Join-Calculus: a Calculus for Distributed Mobile Programming.* PhD thesis, Ecole Polytechnique, Palaiseau, November 1998.

[18] Cédric Fournet and Georges Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of the 23rd ACM Symposium on Principles of Programming Languages*, pages 372–385, January 1996.

[19] Cédric Fournet and Georges Gonthier. A hierarchy of equivalences for asynchronous calculi (extended abstract). In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 844–855. Springer-Verlag, July 1998.

[20] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421. Springer-Verlag, 1996.

[21] Cédric Fournet, Cosimo Laneve, Luc Maranget, and Didier Rémy. Implicit typing à la ML for the join-calculus. In *Proceedings of CONCUR '97*, volume 1243 of *Lecture Notes in Computer Science*, pages 196–212. Springer-Verlag, July 1997.

[22] Cédric Fournet and Luc Maranget. The join-calculus language (version 1.03). Source distribution and documentation available from `http://join.inria.fr/`, June 1997.

[23] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.

[24] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151:437–486, 1995.

[25] Leslie Lamport. A simple approach to specifying concurrent systems. *Communications of the ACM*, 32(1):32–45, January 1989.

[26] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.

[27] Pat Lincoln, John Mitchell, Mark Mitchell, and Andre Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 112–121, November 1998.

[28] John Linn. Generic interface to security services. *Computer Communications*, 17(7):476–482, July 1994.

[29] Jonn Linn. RFC 1508: Generic security service application program interface. Web page at `ftp://ds.internic.net/rfc/rfc1508.txt`, September 1993.

[30] Daniel L. McDonald, Bao G. Phan, and Randall J. Atkinson. A socket-based key management API (and surrounding infrastructure). In *Proceedings of INET'96*, 1996. On the Web at `http://www.isoc.org/isoc/whatis/conferences/inet/96/proceedings/`.

[31] John McLean. Security models. In John Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley & Sons, 1994.

[32] John McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–66, January 1996.

[33] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[34] Microsoft. CryptoAPI. Web pages at `http://www.microsoft.com/security/tech/`.

[35] Robin Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[36] Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2:119–141, 1992.

[37] Robin Milner. The polyadic π-calculus: a tutorial. In Bauer, Brauer, and Schwichtenberg, editors, *Logic and Algebra of Specification*. Springer-Verlag, 1993.

[38] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100:1–40 and 41–77, September 1992.

[39] V. Natarajan and Rance Cleaveland. Divergence and fair testing. In *Proceedings of the 22nd International Colloquium on Automata, Languages and Programming*, volume 944 of *Lecture Notes in Computer Science*, pages 648–659. Springer-Verlag, 1995.

[40] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[41] Davide Sangiorgi and Robin Milner. The problem of "weak bisimulation up to". In *Proceedings of CONCUR '92*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 1992.

[42] Sun Microsystems, Inc. RMI enhancements. Web pages at `http://java.sun.com/products/jdk/1.2/docs/guide/rmi/`, 1997.

[43] Leendert van Doorn, Martín Abadi, Mike Burrows, and Edward Wobber. Secure network objects. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 211–221, May 1996.

[44] Ann Wollrath, Roger Riggs, and Jim Waldo. A distributed object model for the Java system. *Computing Systems*, 9(4):265–290, Fall 1996.