A Top-Down Look at a Secure Message

Martín Abadi[1], Cédric Fournet[2], and Georges Gonthier[3⋆]

[1] Bell Labs Research, Lucent Technologies
[2] Microsoft Research
[3] INRIA Rocquencourt

Abstract. In ongoing work, we are investigating the design of secure distributed implementations of high-level process calculi (in particular, of the join-calculus). We formulate implementations as translations to lower-level languages with cryptographic primitives. Cryptographic protocols are essential components of those translations. In this paper we discuss basic cryptographic protocols for transmitting a single datum from one site to another. We explain some sufficient correctness conditions for these protocols. As an example, we present a simple protocol and a proof of its correctness.

1 Introduction

In the last few years, the scope of security protocols has grown, and so has their complexity. In addition to basic functions such as authentication and key establishment, recent protocols sometimes support elaborate transactions. They may comprise preliminary negotiations, where the parties discuss their preferences and expectations, and layers for application records and for error messages (e.g., [15, 17]). Correspondingly, research on the analysis of security protocols has started to address the challenges of those sophisticated protocols. Examples of this line of work include the recent analyses of the SSL, TLS, IKE, and SET protocols (e.g., [32, 29, 28, 23, 22, 8]).

These trends notwithstanding, in this paper we consider only basic protocols with an elementary goal. This goal is to transmit a single datum from one site to another. The protocols consist of one or more lower-level messages. They employ encryption in order to guarantee the integrity and secrecy of the datum, and nonces or other tags in order to protect against replay attacks.

Despite their simplicity, these protocols serve as building blocks for complex systems. Relying on these protocols, we can add cryptographic protection to an arbitrary program, much as is done in systems with remote invocation facilities [7]. More precisely, we can translate from a process calculus with primitive secure channels (the join-calculus [11]) to a lower-level process calculus where communication across sites may take place on public channels and may use cryptography for security. We have studied such translations in recent papers [3, 4]. The main purpose of this paper is to show a close-up of an essential part of those translations.

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

In addition, these protocols provide an example of a helpful top-down approach to the specification and verification of security protocols. Following this approach, we reduce the problem of implementing the join-calculus to that of writing protocols for transmitting a single datum. We isolate a handful of crisp correctness conditions on the protocols. These conditions are sufficient for the overall correctness of the implementation of the join-calculus. They are not strictly necessary, but each of them corresponds to a sensible requirement on protocols, they are hard to weaken, and they can all be met. We also consider the application of these conditions to specific protocols. In particular, as a didactic example, we present a new, simple protocol and a proof of its correctness.

The next section discusses some of the problems that the top-down approach is intended to address. Section 3 presents the join-calculus and some extensions. Section 4 describes our correctness conditions for protocols. The correctness conditions appear in [3]; the aim of this section is to review them and to explain their implications. Section 5 shows our new protocol; section 6 contains the corresponding proofs. Section 7 concludes. Thus, this paper is partly a review; its main novelties are in informal explanations and in sections 5 and 6.

2 A top-down approach

Work on protocols is seldom purely top-down. Protocol designers often proceed bottom-up, building systems from cryptographic algorithms and network services. Indeed, some protocols arise as applications of cryptographic primitives. For example, Ylönen designed version 1 of the popular protocol SSH [33] as an exercise in the use of RSA; the design of version 2 was "more or less top-down" [34]. Therefore, a top-down approach, on its own, is probably unrealistic.

With this caveat, a top-down approach can serve as a guide. In particular, it helps in addressing common confusions about security protocols and their goals (e.g., [16]). These confusions often enable "attacks" (scenarios that violate some of the expected security properties of a protocol). Although some attacks reveal serious flaws in protocols, many alleged attacks are merely the annoying result of poor protocol specifications, or of poor understandings of those specifications. A top-down perspective helps in distinguishing the dangerous attacks from the unimportant ones, and in avoiding the former.

Those confusions arise even in the analysis of elementary protocols for key exchange. For example, suppose that, after running a certain protocol, two parties A and B are supposed to share a session key. Suppose further that an attacker can arrange that A and B end up with two different keys, but does not know those keys. Surely, this scenario violates the key-establishment goal of the protocol. On the other hand, the scenario may be harmless. When A and B attempt to communicate after running the protocol, they are most likely to discover the discrepancy if their subsequent encrypted messages contain some checkable redundancy. Then A and B are likely to rerun the protocol, and eventually they may agree on a key. At worst, the attack will result in a loss of liveness, but not a loss of safety. (Lowe has considered other attacks with similar properties [19].)

Those confusions become more delicate for current, complex protocols. For example, suppose that A and B start out by discussing which cryptosystem to adopt, using some cleartext messages. Suppose further that an attacker can tamper with this negotiation, convincing A and B to use a cryptosystem of its choice. If A and B adjust the contents of the subsequent conversation to the outcome of the negotiation, then this attack may entail only a loss of liveness: A and B may not send certain sensitive messages. However, this attack could also cause the inappropriate use of a poor cryptosystem, and failures of authenticity and secrecy (e.g., [32]).

Characteristically, these problematic scenarios are vague on the context in which a protocol operates, and on the use of the protocol—for example, how retries happen, or what sorts of application data are sent after a key of a certain type is established. The difficulties diminish or disappear if the protocol is seen as part of a larger system. An attack on the protocol is significant only if it has an effect on the behavior of the larger system.

For our purposes, the larger system is an arbitrary join-calculus program (or a lower-level translation of this program). Top-down, we go from the semantics of the join-calculus to the design of particular protocols; if a protocol is correct according to our conditions then it can serve as a building block for the implementation of the join-calculus. Every potential attack on a protocol can be assessed against the conditions, unambiguously.

In a more restricted view, the larger system could be one that serves a specific purpose, for example a file system, rather than an arbitrary join-calculus program. This specialization may allow more efficient protocols; it also gives rise to the danger that the protocols will be used in unintended, inappropriate ways.

A top-down perspective is not unique to our work. In particular, recent papers by Bellare, Canetti, and Krawczyk [6] and Lynch [21] concern modular methods for designing and analyzing protocols. Although those works are largely disjoint from ours in techniques and results, they seem to have similar conceptual basis and potential benefits. A salient characteristic of our work is that it treats cryptographic operations as black boxes (cf. [6, 18]). A further refinement would be to replace the black boxes with particular cryptographic algorithms. While this refinement is a natural continuation of our approach, it may be quite hard, and we have yet to study it.

3 The join-calculus and its extensions

Next we describe the join-calculus and some additions for representing cryptographic operations and a public network. This review is brief; we refer the reader to previous papers for details on the join-calculus, its theory, and its applications [11, 12, 14, 13, 10, 3, 4].

3.1 The join-calculus

In the join-calculus, processes communicate through named, one-way channels. Intuitively, the channels of the join-calculus have built-in security properties:

- As in the pi-calculus [26, 27], the name of a channel is a transferrable but unforgeable capability.
- A process that sends a message on a channel must have the name of the channel.
- Only the process that creates a channel can receive messages on the channel.

We use lowercase identifiers x, y, *foo*, *bar* ... for names. In addition to a category of names, the syntax of the join-calculus includes categories of values, processes, definitions, and join-patterns. These are defined in the following grammar, where we write $\widetilde{v}$ for a tuple of values $v_1, v_2, \ldots, v_n$.

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

Processes have the following informal semantics:

- $x\langle\widetilde{v}\rangle$ sends the tuple of values $\widetilde{v}$ on x, asynchronously.
- $\mathsf{def}\ D\ \mathsf{in}\ P$ is the process P with the local definitions given in D. In $\mathsf{def}\ D\ \mathsf{in}\ P$, the channel names defined in D are recursively bound in the whole of $\mathsf{def}\ D\ \mathsf{in}\ P$, with lexical scoping.
- $\mathsf{if}\ v = v'\ \mathsf{then}\ P\ \mathsf{else}\ P'$ tests whether $v = v'$, then runs the process P or the process P' depending on the result of the test.
- $P \mid P'$ is the parallel composition of the processes P and P'.
- $\mathsf{0}$ is the null process, which does nothing.

A definition $J \triangleright P$ says that the process P may run when there are messages that match the join-pattern J. For example, let D be the definition:

$$(x_1\langle y_1\rangle \mid x_2\langle y_2, y_3\rangle) \triangleright (x_1\langle y_1\rangle \mid z\langle y_1, y_2, y_3\rangle)$$

This definition introduces two channels, with names x_1 and x_2. The names y_1, y_2, and y_3 are also bound; they are formal parameters that may be instantiated to actual values received on x_1 and x_2. The name z is free. When a message $x_1\langle v_1\rangle$ appears on x_1 and a message $x_2\langle v_2, v_3\rangle$ appears on x_2, this definition may fire, consuming both messages, reproducing $x_1\langle v_1\rangle$, and producing $z\langle v_1, v_2, v_3\rangle$. Thus, $\mathsf{def}\ D\ \mathsf{in}\ x_1\langle v_1\rangle \mid x_2\langle v_2, v_3\rangle$ yields $\mathsf{def}\ D\ \mathsf{in}\ x_1\langle v_1\rangle \mid z\langle v_1, v_2, v_3\rangle$.

3.2 Syntactic extensions

It is convenient to introduce some syntactic extensions that do not affect the expressiveness of the calculus. We write $\mathsf{repl}\ P$ for the replication of P, which can be defined as $\mathsf{def}\ x\langle\rangle \triangleright (P \mid x\langle\rangle)\ \mathsf{in}\ x\langle\rangle$ for a fresh x. In addition, we have notations for data structures that contain unique identifiers. We write the declaration $\mathsf{uids}\ t$ as the definition of an initially empty set t; and write $\mathsf{if\ not\ tset}\ t(c)\ \mathsf{then}\ P$ for a process that atomically tests whether 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.)

3.3 Cryptographic primitives

We define the sjoin-calculus, which is analogous to the spi-calculus [5], by enriching the join-calculus with a few constructs:

- $\mathsf{fresh}\ x$ is a definition that introduces the fresh name x; this name may for example be included as a unique identifier within a message.
- $\mathsf{keys}\ x^+, x^-$ is a definition that introduces a pair of keys, for public-key encryption [24]; x^+ is an encryption key (the "public key") and x^- is the inverse decryption key (the "secret key").
- $\{\widetilde{v}\}_v$ is a value that represents the result of encrypting $\widetilde{v}$ with key v. The inverse of v should be used for decryption.
- $\mathsf{decrypt}\ v\ \mathsf{using}\ v'\ \mathsf{to}\ \widetilde{x}\ \mathsf{in}\ P\ \mathsf{else}\ P'$ is a process that attempts to decrypt v using v' as key. If the decryption succeeds, then P runs, with the results of the decryption substituted for $\widetilde{x}$; otherwise, P' runs. We may omit P' when it is $\mathsf{0}$.

With these additions, we can represent systems where low-level processes use cryptography for security.

3.4 A public network

In addition, we need a model of a public, asynchronous network over which low-level processes communicate. This network should allow an attacker to intercept, modify, duplicate, and inject messages. Sometimes we may also wish to assume that the attacker has access to certain public keys without knowing the corresponding secret keys. Therefore, we define the contexts:

$$\begin{aligned}
\mathcal{E}\,[\,\cdot\,] &\stackrel{\text{def}}{=} \mathsf{def}\ \mathit{recv}\langle\kappa\rangle \mid \mathit{emit}\langle m\rangle \triangleright \kappa\langle m\rangle\ \mathsf{in}\ [\,\cdot\,] \\
\mathcal{E}\mathit{nv}\,[\,\cdot\,] &\stackrel{\text{def}}{=} \mathcal{E}\,[\,\mathit{plug}\langle \mathit{emit}, \mathit{recv}\rangle \mid W \mid \cdot\,] \\
\mathcal{E}\mathit{nv}_{x^\pm}\,[\,\cdot\,] &\stackrel{\text{def}}{=} \mathsf{def\ keys}\ x^+, x^-\ \mathsf{in}\ \mathcal{E}\mathit{nv}\left[\,\mathit{plug}'\langle x^+\rangle \mid \cdot\,\right]
\end{aligned}$$

where:

- emit and recv represent the network interface. For output, a process sends its message on emit. For input, it sends a continuation channel κ on recv, and the network may return a message on κ.

– $plug$ and $plug'$ are auxiliary channels whose sole purpose is to make certain public names ($emit$, $recv$, x^+) available to outside processes.
– W is the process $\mathsf{repl}\ (\mathsf{def\ fresh}\ m\ \mathsf{in\ repl}\ emit\langle m\rangle)$, which repeatedly puts fresh messages on the network, as background noise, helping protect against traffic-analysis attacks.

Intuitively, $\mathcal{E}nv\,[\,\cdot\,]$ represents the network; $\mathcal{E}nv\,[\,P\,]$ describes a situation where both a process P and any process running in parallel may use $emit$ and $recv$. The other contexts play auxiliary roles.

3.5 Operational semantics and types (notation)

The operational semantics of the join-calculus and its extensions define the following notations and concepts [2].

– $P \downarrow_x$ holds if P may output on x immediately. For example, $x\langle v\rangle \downarrow_x$.
– $P \Downarrow_x$ holds if P may output on x either immediately or after some internal reductions. For example, $(\mathsf{def}\ y\langle\rangle \triangleright x\langle\rangle\ \mathsf{in}\ y\langle\rangle) \Downarrow_x$.
– $P \equiv Q$ holds if P and Q are structurally equivalent, that is, if P and Q differ only by certain simple syntactic rearrangements. For example, for all P_1 and P_2, $P_1 \mid P_2 \equiv P_2 \mid P_1$. We also take $\mathsf{repl}\ P \equiv (P \mid \mathsf{repl}\ P)$.
– $P \rightarrow Q$ holds if P may reduce to Q, that is, if P may perform one step of internal computation and then behave as Q. For example,

$$\mathsf{def}\ y\langle\rangle \triangleright x\langle\rangle\ \mathsf{in}\ y\langle\rangle \ \rightarrow\ \mathsf{def}\ y\langle\rangle \triangleright x\langle\rangle\ \mathsf{in}\ x\langle\rangle$$

The relations $\rightarrow^=$ and $\rightarrow^*$ are the reflexive closure and the reflexive-transitive closure of $\rightarrow$, respectively.
– An evaluation context $C[\,\cdot\,]$ is a context in which computation may immediately take place: if $P \rightarrow Q$ then $C[P] \rightarrow C[Q]$. For example, $P_1 \mid \cdot$ is an evaluation context, while $\mathsf{decrypt}\ v\ \mathsf{using}\ v'\ \mathsf{to}\ \widetilde{x}\ \mathsf{in}\ \cdot\ \mathsf{else}\ P_2$ is not.

We rely on simple, monomorphic type systems for the calculi. In these type systems, each channel has an associated arity (an integer size for the tuples of values transmitted on the channel). We write $\langle \tau_1, \ldots, \tau_n\rangle$ for the type of channels that carry tuples with n values of respective types $\tau_1, \ldots, \tau_n$. We allow types to be recursively defined (formally, using a fixpoint operator), so we may have for example $\tau = \langle \tau, \tau\rangle$.

In addition to channel types, we have a basic type $\mathsf{BitString}$. This is the type of keys, ciphertexts and their contents, and of the fresh names introduced with the construct $\mathsf{fresh}\ \cdot$.

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.

4 Correctness

In this section we arrive, more or less top-down, at the problem of transmitting a datum securely. We present correctness conditions for protocols that solve the problem.

4.1 Goals

Now we can write both pure join-calculus processes, where security is based on the scoping of channel names, and processes that communicate using cryptography over a public network (as represented by $\mathcal{E}nv\,[\,\cdot\,]$). Our objective is to see the latter as implementations of the former. Moreover, we wish to do this systematically: we would like to have compilers that map pure join-calculus processes to lower-level code that can execute securely over a public network.

In order to formalize the security requirement for such compilers, we resort to process equivalences. We say that two processes P_1 and P_2 are equivalent, and write $P_1 \approx P_2$, when no context can distinguish one from the other [25, 9]. Intuitively, we may view the context as an attacker; then $P_1 \approx P_2$ entails an integrity property (limiting the effect of the attacker on the behavior of P_1 and P_2) and a secrecy property (limiting the observations of the attacker). Formally, we define $\approx$ as the largest symmetric relation $\mathcal{R}$ on processes such that:

1. if $P \mathrel{\mathcal{R}} Q$ and $P \Downarrow_x$ then $Q \Downarrow_x$;
2. $\mathcal{R}$ is a congruence for all 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, if $P \mathrel{\mathcal{R}} Q$ and $P \rightarrow^* P'$ then, for some Q', $P' \mathrel{\mathcal{R}} Q'$ and $Q \rightarrow^* Q'$.

When we devise implementations of the join-calculus, we wish to preserve equivalences, since equivalences can express security properties [5, 1]. More precisely, if P_1 and P_2 are equivalent join-calculus processes, we wish to compile them to processes P'_1 and P'_2 such that $\mathcal{E}nv\,[\,P'_1\,]$ and $\mathcal{E}nv\,[\,P'_2\,]$ are equivalent. The mention of $\mathcal{E}nv\,[\,\cdot\,]$ accounts for the use of the public network by P'_1 and P'_2.

4.2 Protocols

A crucial part of implementing the join-calculus is mapping join-calculus messages to cryptographic protocols. Following an obvious strategy and postponing optimizations, we associate a pair of keys x^+, x^- with each join-calculus channel x.

- The encryption key x^+ corresponds to the capability of sending messages on x, which may be transferred.
- The decryption key x^- corresponds to the capability of receiving messages on x, which only the creator of x has.

In order to simulate communication on a join-calculus channel x, we employ protocols that consist of two processes $E_x[\widetilde{v}]$ and R_x, for sending and receiving, respectively.

- Using the key x^+ for encryption, $E_x[\widetilde{v}]$ sends $\widetilde{v}$. Within $\widetilde{v}$, an encryption key y^+ (rather than a pair y^+, y^-) represents the corresponding channel y.
- Using the key x^- for decryption, R_x receives messages, then forwards their cleartext contents on an auxiliary, internal channel x°.

For example, a protocol might be:

$$\begin{aligned} E_x[\widetilde{v}] &\stackrel{\text{def}}{=} \ emit\langle\{\widetilde{v}\}_{x^+}\rangle \\ R_x &\stackrel{\text{def}}{=} \ \mathsf{def}\ \kappa\langle m\rangle \triangleright \mathsf{decrypt}\ m\ \mathsf{using}\ x^-\ \mathsf{to}\ \widetilde{y}\ \mathsf{in}\ x^\circ\langle\widetilde{y}\rangle \\ &\quad\ \ \mathsf{in}\ recv\langle\kappa\rangle \end{aligned}$$

where the length of $\widetilde{y}$ is deduced from the type of x. For instance, using this protocol, the join-calculus process $x\langle y\rangle$ may be mapped to $emit\langle\{y^+\}_{x^+}\rangle$. This naive protocol is subject to message replays and other obvious attacks. The protocol of section 5 thwarts those attacks.

In general, such protocols should guarantee the integrity and secrecy of $\widetilde{v}$. These properties are informally appealing, and they are formally necessary for the desired preservation of equivalences. Section 4.3 gives a more precise and complete list of properties.

4.3 Correctness conditions for protocols

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 an expansion relation on processes ($\succeq$) [31], which is similar to $\approx$ but stronger and asymmetric. We define $\succeq$ as the largest relation $\mathcal{R}$ such that $\mathcal{R}$ and its converse meet requirements (1), (2), and (3) of the definition of $\approx$, and such that if $P\ \mathcal{R}\ Q$ and $P \rightarrow P'$ then, for some Q', $P'\ \mathcal{R}\ Q'$ and $Q \rightarrow^= Q'$.

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 the protocol $(\mathsf{0}, x^\circ\langle\cdot\rangle)$. 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 does not send messages on *emit* or x° spontaneously. 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. Thus, this condition implies that the state of the receiving process remains essentially unchanged as long as it interacts with regular sending processes; any state change in the receiving process, such as the addition of entries in internal tables, should not be observable. (Condition 4 is slightly stronger in [3], where the assumption that x^- does not occur in $\widetilde{v}$ is missing.)

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 4.2 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^+ and 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 $\mathcal{E}nv_{x^\pm}\,[\,R \mid emit\langle v\rangle\,] \to P$ 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 protocol of section 5), 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 $\mathcal{E}nv_{x^\pm}\,[\,R \mid emit\langle\{\widetilde{v}\}_{x^+}\rangle\,] \to P$ 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.

As mentioned above, these correctness conditions suffice for our results about the implementations of the join-calculus. Compared to common specifications of cryptographic protocols (e.g., [20]), our correctness conditions are rather extensional [30]: they concern the intended effects of a protocol rather than its internal behavior.

5 A protocol

In this section, we describe a specific protocol for transmitting a single datum. This protocol is an instructive example: it is simple and correct; moreover, its correctness is not too hard to prove. On the other hand, this protocol is not very practical. Two other correct protocols appear in [3]; they are somewhat more complex and realistic. In current work (discussed briefly in section 7), we are making further efficiency improvements.

Our simple protocol is an enhancement of the naive protocol in several respects:

- Each encrypted message contains a fresh component c, which serves as a confounder (making the message unpredictable and different from other messages with the same payload) and as a unique identifier (against replay attacks).
- Each encrypted message is repeated indefinitely, in case some copies are intercepted.
- When a process receives a message for the first time, it reemits the message and records it before further processing. Later, it reemits but ignores duplicates of this message.

The first correct protocol of [3] is similar in these respects, but it does fewer reemissions and uses records of unique identifiers instead of records of complete messages.

Since calls to *recv* retrieve messages from the network non-deterministically, and since messages may be duplicated by the sender or by an attacker, processes have to filter for messages that are destined for them. This filtering relies first on a table for discarding duplicates, then on a decryption key for accessing message contents. We arrive at the following definitions.

$$E_x[\widetilde{v}] \stackrel{\text{def}}{=} \mathsf{def\ fresh}\ c\ \mathsf{in\ repl}\ emit\langle\{c,\widetilde{v}\}_{x^+}\rangle$$

$$\begin{aligned}
R_x &\stackrel{\text{def}}{=} R_x\{\}\,[0] \\
R_x\{\widetilde{u}\}\,[P] &\stackrel{\text{def}}{=} \mathsf{def\ uids}\ t_x\{\widetilde{u}\}\ \mathsf{in} \\
&\qquad \mathsf{def}\ \kappa\langle m\rangle \triangleright emit\langle m\rangle \mid F_{m,x}\ \mathsf{in}\ P \mid \mathsf{repl}\ recv\langle\kappa\rangle \\
F_{m,x} &\stackrel{\text{def}}{=} \mathsf{if\ not\ tset}\ t_x(m)\ \mathsf{then}\ F'_{m,x} \\
F'_{m,x} &\stackrel{\text{def}}{=} \mathsf{decrypt}\ m\ \mathsf{using}\ x^-\ \mathsf{to}\ c,\widetilde{y}\ \mathsf{in}\ x^\circ\langle\widetilde{y}\rangle
\end{aligned}$$

where the length of the tuple $\widetilde{y}$ in the definition of $F'_{m,x}$ is deduced from the type of x. In the receiving process, the component $F_{m,x}$ serves as a filter for a single message; such components are replicated, and share a set of previously received messages.

Theorem 1. *The protocol* $(R_x, E_x[\cdot])$ *is correct.*

The proof of this theorem is the subject of the next section

6 A correctness proof

The proof of Theorem 1 relies on a series of lemmas that describes the various stages of communications processing (reception, duplicate elimination, decryption). Proofs for more sophisticated protocols have similar structure, though each of the steps becomes harder.

Throughout, we employ several techniques for establishing equivalences and other relations between processes. These techniques are not specific to cryptography and appear in lemmas in [2]; here, we only present their role informally. We also use the equivalence relation $\asymp$, which is stronger than $\approx$ and is obtained from the definition of $\approx$ in section 4.1 by substituting $\rightarrow^=$ for $\rightarrow^*$. The following "up-to" technique is helpful for proving $\asymp$. In order to show that $\mathcal{R} \subseteq \asymp$, it suffices to prove, for all P and Q such that $P \mathrel{\mathcal{R}} Q$, that:

1. if $P \downarrow_x$ then $Q \Downarrow_x$; conversely, if $Q \downarrow_x$ then $P \Downarrow_x$;
2. $C[P] \asymp\mathcal{R}\asymp C[Q]$ for every context $C[\cdot]$ of the form $\mathsf{def}\ D\ \mathsf{in}\ R \mid [\cdot]$ such that the names bound in P and Q do not occur in D and R;
3. if $P \rightarrow P'$, then $Q \rightarrow^= Q'$ and $P' \asymp\mathcal{R}^=\asymp Q'$ for some Q'; conversely, if $Q \rightarrow Q'$, then $P \rightarrow^= P'$ and $P' \asymp\mathcal{R}^=\asymp Q'$ for some P'.

(Here, $\mathcal{R}^=$ is the reflexive closure of $\mathcal{R}$ and $\asymp\mathcal{R}\asymp$ is its composition with $\asymp$ on both sides.)

We focus on a fixed channel x and the associated names x^+, x^-, and x°. In the remainder of this section, we assume that these names and *emit* and *recv* are never alpha-converted. The up-to proof techniques used in this section allow this assumption.

We say that a value m is a well-formed message when it is of the form $\{c, \widetilde{v}\}_{x^+}$ for some values $c, \widetilde{v}$ of type $\mathsf{BitString}$ such that the length of $\widetilde{v}$ is the arity of x. A process P is an internal state when it is a parallel composition of processes $\kappa\langle m\rangle$, $F_{m,x}$, or $F'_{m,x}$, for some $\mathsf{BitString}$ values m in which x^- does not appear. A net state is a process of the form:

$$\mathcal{E}\left[\,\mathsf{def\ keys}\ x^+, x^- \wedge D\ \mathsf{in}\ R_x\{\widetilde{u}\}\,[P] \mid Q\,\right]$$

where D does not define *emit*, *recv*, x^+, or x^-, the tuple $\widetilde{u}$ contains pairwise distinct $\mathsf{BitString}$ values, P is an internal state, and x^- does not appear in D, Q, or $\widetilde{u}$. For any net state S, there exist a unique context $C[\cdot]$, tuple $\widetilde{u}$, process Q, and internal state P such that $S = C[R_x\{\widetilde{u}\}\,[P] \mid Q]$. Moreover, $\widetilde{u}$ and P can be derived (uniquely up to alpha-conversion and reordering for $\widetilde{u}$ and $\equiv$ for P)

from any $T \equiv S$, because $R_x\{\widetilde{u}\}\,[P]$ must contain all occurrences of x^- and κ in S.

The first lemma says that net states are closed under application of evaluation contexts and under reduction.

Lemma 1 (Closure). *Let S be a net state. For any evaluation context $E[\,\cdot\,]$ in which names bound in S do not appear, we have $E[S] \equiv S'$ for some net state S'. For any reduction step $S \to T$, we have $T \equiv S'$ for some net state S'.*

Proof. Let $S = C\,[R_x\{\widetilde{u}\}\,[P] \mid Q]$, with $C[\,\cdot\,] = \mathcal{E}\,[\,\mathsf{def\ keys}\,x^+, x^- \wedge D\ \mathsf{in}\ \cdot\,]$. For the first claim, it is enough to consider contexts $E[\,\cdot\,]$ of the forms $(\,\cdot \mid Q')$ and $(\mathsf{def}\ D'\ \mathsf{in}\ \cdot\,)$. We obtain S' from S by replacing Q with $Q \mid Q'$ or D with $D \wedge D'$, respectively. For the second claim, we base our analysis on the location of the processes involved in the reduction step, and further on the type of reduction if the reduction is local to $R_x\{\widetilde{u}\}\,[P]$:

- If the step involves only processes in Q (possibly with definitions in $C[\,\cdot\,]$), then it leaves $R_x\{\widetilde{u}\}\,[P]$ unchanged, so there must be D', Q', and $C'[\,\cdot\,]$ such that $C'[\,\cdot\,] = \mathcal{E}\,[\,\mathsf{def\ keys}\,x^+, x^- \wedge D'\ \mathsf{in}\ \cdot\,]$, $T \equiv C'\,[R_x\{\widetilde{u}\}\,[P] \mid Q']$, and $C[Q] \to C'[Q']$. Therefore, x^- does not appear in D' and Q', so we take $S' = C'\,[R_x\{\widetilde{u}\}\,[P] \mid Q']$.
- If the step involves processes in both Q and $R_x\{\widetilde{u}\}\,[P]$, then it must use the rule in $\mathcal{E}\,[\,\cdot\,]$ to produce a message $\kappa\langle m\rangle$ from a message $emit\langle m\rangle$ in Q and a message $recv\langle\kappa\rangle$ unfolded from $R_x\{\widetilde{u}\}\,[\,\cdot\,]$. We can assume that this message $emit\langle m\rangle$ does not appear under any definitions in Q, since any such definition can be moved to D. Hence we have $Q \equiv emit\langle m\rangle \mid Q'$ for some process Q', so we take $S' = C\,[R_x\{\widetilde{u}\}\,[P \mid \kappa\langle m\rangle] \mid Q']$.
- If the step replaces a message $\kappa\langle m\rangle$ in P with $emit\langle m\rangle \mid F_{m,x}$ using the rule in $R_x\{\widetilde{u}\}\,[\,\cdot\,]$, then $P \equiv P' \mid \kappa\langle m\rangle$ for some internal state P', so we take $S' = C\,[R_x\{\widetilde{u}\}\,[P' \mid F_{m,x}] \mid (emit\langle m\rangle \mid Q)]$.
- If the step executes the test-and-set in some process $F_{m,x}$ in P, entering m in the table for x, then $P \equiv P' \mid F_{m,x}$ for some internal state P', so we take $S' = C\left[R_x\{\widetilde{u}, m\}\left[P' \mid F'_{m,x}\right] \mid Q\right]$.
- Otherwise, the step evaluates the decryption in some process $F'_{m,x}$ in P, yielding $x^\circ\langle\widetilde{v}\rangle$ if m is a well-formed message $\{c, \widetilde{v}\}_{x^+}$ and $\mathsf{0}$ otherwise, and $P \equiv P' \mid F'_{m,x}$ for some internal state P', so if $m = \{c, \widetilde{v}\}_{x^+}$ we take $S' = C\,[R_x\{\widetilde{u}\}\,[P'] \mid (x^\circ\langle\widetilde{v}\rangle \mid Q)]$, and otherwise $S' = C\,[R_x\{\widetilde{u}\}\,[P'] \mid Q]$. $\square$

Lemma 2 shows how a receiving process $R_x\{\widetilde{u}\}\,[P]$ incorporates a message from the public network. The message is immediately reemitted, independently of its contents.

Lemma 2 (Reception). *If $C\,[R_x\{\widetilde{u}\}\,[P] \mid (emit\langle m\rangle \mid Q)]$ is a net state, then:*

$$\begin{aligned} C\big[R_x\{\widetilde{u}\}\,[P] \mid (emit\langle m\rangle \mid Q)\big] &\to C\big[R_x\{\widetilde{u}\}\,[\kappa\langle m\rangle \mid P] \mid Q\big] \\ &\succeq C\big[R_x\{\widetilde{u}\}\,[F_{m,x} \mid P] \mid (emit\langle m\rangle \mid Q)\big] \end{aligned}$$

Proof. In more detail, we have:

$$\begin{aligned} C\big[R_x\{\widetilde{u}\}\,[P] \mid (emit\langle m\rangle \mid Q)\big] &\equiv C\big[R_x\{\widetilde{u}\}\,[(emit\langle m\rangle \mid recv\langle \kappa\rangle) \mid P] \mid Q\big] \\ &\to C\big[R_x\{\widetilde{u}\}\,[\kappa\langle m\rangle \mid P] \mid Q\big] \\ &\to C\big[R_x\{\widetilde{u}\}\,[emit\langle m\rangle \mid F_{m,x} \mid P] \mid Q\big] \\ &\equiv C\big[R_x\{\widetilde{u}\}\,[F_{m,x} \mid P] \mid (emit\langle m\rangle \mid Q)\big] \end{aligned}$$

The initial structural equivalence unfolds a copy of the replicated process $recv\langle\kappa\rangle$ within $R_x\{\widetilde{u}\}\,[\cdot]$ and groups this process with $emit\langle m\rangle$. The first reduction step replaces the process $emit\langle m\rangle \mid recv\langle\kappa\rangle$ with the process $\kappa\langle m\rangle$ according to the rule defining $emit$ and $recv$ within $\mathcal{E}[\,\cdot\,]$. The second reduction step consumes $\kappa\langle m\rangle$ according to the rule within $R_x\{\widetilde{u}\}\,[\cdot]$. The final structural equivalence moves $emit\langle m\rangle$ back to its original position.

The second step depends only on the deterministic definition of κ. Using a standard lemma of the join-calculus, we can substitute $\succeq$ for $\to$ in this case. ◻

Similarly, Lemma 3 deals with a replicated message. In this case, one copy of the message always remains available, so the first reduction of Lemma 2 is also an expansion.

Lemma 3 (Reception of a replicated message). *Let $C'[\cdot]$ be a context of the form $C\left[R_x\{\widetilde{u}\}\,[\cdot \mid P] \mid \big((\mathsf{repl}\ emit\langle m\rangle) \mid Q\big)\right]$, such that $C'[\mathsf{0}]$ is a net state. Then $C'[\mathsf{0}] \succeq C'[F_{m,x}]$.*

Proof. We have:

$$\begin{aligned} C'[\mathsf{0}] &\equiv C'[emit\langle m\rangle] \\ &\to C'[\kappa\langle m\rangle] \\ &\to C'[emit\langle m\rangle \mid F_{m,x}] \\ &\equiv C'[F_{m,x}] \end{aligned}$$

The first structural equivalence unfolds a copy of the replicated message $emit\langle m\rangle$. The two reduction steps apply the rules for $emit, recv$ and κ, respectively, as in Lemma 2. The final structural equivalence folds back the copy of $emit\langle m\rangle$.

Next, we show that the reduction sequence described above is also an expansion. Given some $\mathsf{BitString}$ value m, we let $\mathcal{R}$ be the relation that contains all the pairs of processes $C'[\mathsf{0}], C'[F_{m,x}]$ related by the lemma. We show that $\mathcal{R} \subseteq \succeq$. By construction, we have $\mathcal{R} \subseteq \to^2$, so we can apply the special case of the expansion-up-to-expansion proof technique (Lemma 3 of [2]) to analyze two processes $C'[\mathsf{0}]$ and $C'[F_{m,x}]$ such that $C'[\mathsf{0}]\ \mathcal{R}\ C'[F_{m,x}]$. That proof technique requires matching the reduction steps of $C'[\mathsf{0}]$ with those of $C'[F_{m,x}]$, and showing that $\mathcal{R}$ is closed under application of an evaluation context $E[\cdot]$ in which variables bound in $C'[\mathsf{0}]$ or $C'[F_{m,x}]$ do not appear. Every step $C'[\mathsf{0}] \to T$ can obviously be matched by a step of $C'[F_{m,x}]$, since the inert term $\mathsf{0}$ cannot take part in the step: there is a context $C''[\cdot]$ such that $T \equiv C''[\mathsf{0}]$ and $C'[F_{m,x}] \to C''[F_{m,x}]$. By Lemma 1 we can choose $C''[\cdot]$ so that $C''[F_{m,x}]$ is a net state, whence $C''[\mathsf{0}]$ is one as well. The term $\mathsf{repl}\ emit\langle m\rangle$ cannot disappear in the step, so it must occur in the 'Q' part of $C''[\cdot]$. As in the proof of Lemma 1, we can choose $C''[\cdot]$ so that this part has the form $(\mathsf{repl}\ emit\langle m\rangle) \mid Q'$, so $C''[\mathsf{0}]\ \mathcal{R}\ C''[F_{m,x}]$. A similar argument shows that $\mathcal{R}$ is closed under application of evaluation contexts. $\square$

Lemmas 4 and 5 describe the following steps of communications processing. They concern the freshness of a message and its well-formedness, respectively.

Lemma 4 (Duplicates). *If $C\left[R_x\{\widetilde{u}\}\left[P\right]\right]$ is a net state, then:*

$$C\left[R_x\{\widetilde{u}\}\left[P \mid F_{m,x}\right]\right] \succeq C\left[R_x\{\widetilde{u},m\}\left[P \mid F'_{m,x}\right]\right] \tag{1}$$
$$C\left[R_x\{\widetilde{u},m\}\left[P \mid F_{m,x}\right]\right] \asymp C\left[R_x\{\widetilde{u},m\}\left[P\right]\right] \tag{2}$$

Proof. This follows from the definition of tables of unique identifiers, standard up-to proof techniques, and Lemma 1. In the first relation, there may be other copies of $F_{m,x}$ in P attempting to enter m in the table for x, but each copy yields the same process $F'_{m,x}$, so the choice of a particular copy is not observable. $\square$

Lemma 5 (Decryption). *If $C[F'_{m,x}]$ is a net state, then $C[F'_{m,x}] \succeq C[x^\circ\langle\widetilde{v}\rangle]$ when m is a well-formed message $\{c,\widetilde{v}\}_{x^+}$ and $C[F'_{m,x}] \asymp C[\mathsf{0}]$ otherwise.*

Proof. This follows from the definition of decryption in the sjoin-calculus, standard up-to proof techniques, and Lemma 1. $\square$

Composing these last three lemmas in the case where there is an emitter $E_x[\widetilde{v}]$, we can summarize a successful run of the protocol as follows:

Lemma 6 (Completion). *If $S = C\left[R_x\{\widetilde{u}\}\left[P\right] \mid \left(E_x[\widetilde{v}] \mid Q\right)\right]$ is a net state, c is a* BitString *name that does not appear in S, and $m = \{c,\widetilde{v}\}_{x^+}$ is a well-formed message, then:*

$$\begin{aligned} S &\equiv \mathsf{def\ fresh}\ c\ \mathsf{in}\ C\big[R_x\{\widetilde{u}\}\left[P\right] \mid \left(\left(\mathsf{repl}\ emit\langle m\rangle\right) \mid Q\right)\big] \\ &\succeq \mathsf{def\ fresh}\ c\ \mathsf{in}\ C\big[R_x\{\widetilde{u},m\}\left[P\right] \mid \left(x^\circ\langle\widetilde{v}\rangle \mid \left(\mathsf{repl}\ emit\langle m\rangle\right) \mid Q\right)\big] \end{aligned}$$

Proof. We simply unfold the definitions, use structural equivalence, and successively apply Lemmas 3, 4 (first part), and 5 (first case). The first structural equivalence 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, so there is no capture as its definition is lifted outside. As we apply Lemma 4, the replicated message is different from any value appearing in the table for x because no such value may contain c. $\square$

Lemmas 7 and 8 state that what is left after a successful run of the protocol is indistinguishable from noise, and can be discarded up to $\asymp$. Lemma 7 says that a well-formed message m that appears in the table for a channel x can be uniformly replaced with a fresh name d. Lemma 8 says that an ill-formed value can be discarded from the table.

Lemma 7 (Noise). *If $S = C\,[R_x\{\widetilde{u}, d\}\,[P]]$ is a net state, d is a* BitString *name that is not bound in S, $F'_{d,x}$ does not appear unguarded in P, c is a* BitString *name that does not appear in S, and $m = \{c, \widetilde{v}\}_{x^+}$, then:*

$$\mathsf{def\ fresh}\ c\ \mathsf{in}\ \left(S\{^m/_d\}\right) \asymp \mathsf{def\ fresh}\ d\ \mathsf{in}\ S$$

Proof. We let $\mathcal{R}$ be the relation that contains all pairs of processes equated in the lemma, and prove that $\mathcal{R} \subseteq \asymp$. Every reduction commutes with the substitution; in particular:

- the substitution $\{^m/_d\}$ is an injection from values in which c does not appear to values in which d does not appear, so every reduction step that depends on the comparison of two values selects the same branch before and after the substitution;
- S does not attempt to decrypt d with x^-, and $S\{^m/_d\}$ does not attempt to decrypt m with x^-, since $F'_{d,x}$ does not appear unguarded in P and x^- does not appear elsewhere in S.

By Lemma 1, if $S \to S'$ then we can take S' to be a net state. Furthermore, $F'_{d,x}$ cannot appear unguarded in S': the processes $F_{d,x}$ that may appear in P are inert, because d already appears in the table. It follows that $\mathcal{R}$ is closed under reduction. Closure under application of evaluation contexts follows from Lemma 1. Finally, $\{^m/_d\}$ does not operate on channel names, so for every channel y we have $S \downarrow_y$ if and only if $S\{^m/_d\} \downarrow_y$. □

Lemma 8 (Table simplification). *If $C\,[R_x\{\widetilde{u}, v\}\,[P]]$ is a net state where v is not a well-formed message, then $C\,[R_x\{\widetilde{u}, v\}\,[P]] \asymp C\,[R_x\{\widetilde{u}\}\,[P]]$.*

Proof. We let $\mathcal{R}$ be the relation that contains all pairs of processes equated in the lemma, and prove that $\mathcal{R} \subseteq \asymp$. To establish the bisimulation requirement, we compare the reductions of processes that $\mathcal{R}$ relates. In the case where $P \equiv F_{v,x} \mid P'$ for some internal state P', a reduction step is enabled only on the right-hand side:

$$C\,[R_x\{\widetilde{u}\}\,[F_{v,x} \mid P']] \to C\left[R_x\{\widetilde{u}, v\}\left[F'_{v,x} \mid P'\right]\right]$$

The left-hand side does not need to match this step, as the second part of Lemma 4 and the second case of Lemma 5 give us:

$$C\,[R_x\{\widetilde{u}, v\}\,[F_{v,x} \mid P']] \asymp C\,[R_x\{\widetilde{u}, v\}\,[P']] \asymp C\left[R_x\{\widetilde{u}, v\}\left[F'_{v,x} \mid P'\right]\right]$$

In all other cases, reductions are the same on both sides, and lead to related net states by Lemma 1. Closure under application of evaluation contexts is likewise a direct consequence of Lemma 1. Finally, immediate outputs are the same on both sides of $\mathcal{R}$. □

We are now ready to prove the correctness requirements for our protocol (Theorem 1):

Proof. We let $\mathbf{R}$ consist of all terms $R_x\{\widetilde{u}\}\,[\mathsf{0}]$ where $\widetilde{u}$ is a tuple of pairwise distinct values of type $\mathsf{BitString}$ in which x^- does not appear. Thus, $\mathbf{R}$ consists of derivatives of R_x with different contents in the table of unique identifiers.

1: By definition, $R_x = R_x\{\}\,[\mathsf{0}]$.

2, 3: These conditions are syntactically obvious for $E_x[\,\cdot\,]$ and for every $R \in \mathbf{R}$.

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 $R \in \mathbf{R}$ and every tuple $\widetilde{v}$ of $\mathsf{BitString}$ values whose length matches the arity of x and in which x^- does not appear. We derive this expansion relation by composing Lemmas 6, 7, and 8. Let $R = R_x\{\widetilde{u}\}\,[\mathsf{0}]$, let c and d be fresh $\mathsf{BitString}$ names, and let $m = \{c, \widetilde{v}\}_{x^+}$; we have:

$$\begin{aligned}
&\mathcal{E}nv_{x^\pm}\,[\,R_x\{\widetilde{u}\}\,[\mathsf{0}] \mid E_x[\widetilde{v}]\,] \\
\succeq\;& \mathsf{def\ fresh}\ c\ \mathsf{in}\ \mathcal{E}nv_{x^\pm}\,[\,R_x\{\widetilde{u}, m\}\,[\mathsf{0}] \mid (\mathsf{repl}\ emit\langle m\rangle) \mid x^\circ\langle\widetilde{v}\rangle\,] && (1)\\
\asymp\;& \mathsf{def\ fresh}\ d\ \mathsf{in}\ \mathcal{E}nv_{x^\pm}\,[\,R_x\{\widetilde{u}, d\}\,[\mathsf{0}] \mid (\mathsf{repl}\ emit\langle d\rangle) \mid x^\circ\langle\widetilde{v}\rangle\,] && (2)\\
\asymp\;& \mathsf{def\ fresh}\ d\ \mathsf{in}\ \mathcal{E}nv_{x^\pm}\,[\,R_x\{\widetilde{u}\}\,[\mathsf{0}] \mid (\mathsf{repl}\ emit\langle d\rangle) \mid x^\circ\langle\widetilde{v}\rangle\,] && (3)\\
\equiv\;& \mathcal{E}nv_{x^\pm}\,[\,R_x\{\widetilde{u}\}\,[\mathsf{0}] \mid x^\circ\langle\widetilde{v}\rangle\,] && (4)
\end{aligned}$$

Let $Q_0 = W \mid plug\langle emit, recv\rangle \mid plug'\langle x^+\rangle$. The expansion (1) is obtained by applying Lemma 6, taking $C[\,\cdot\,] = \mathcal{E}\,[\,\mathsf{def\ keys}\ x^+, x^-\ \mathsf{in}\ \cdot\,]$ and $Q = Q_0$. The relations (2) and (3) are obtained by applying Lemmas 7 and 8, respectively, taking $C[\,\cdot\,] = \mathcal{E}\,[\,\mathsf{def\ keys}\ x^+, x^-\ \mathsf{in}\ \cdot \mid (x^\circ\langle\widetilde{v}\rangle \mid (\mathsf{repl}\ emit\langle d\rangle) \mid Q_0)\,]$. The structural equivalence (4) is obtained by first restricting the scope of d to the process $\mathsf{repl}\ emit\langle d\rangle$—by hypothesis, d does not occur elsewhere—then by using the structural equivalence $W \equiv W \mid \mathsf{def\ fresh}\ d\ \mathsf{in\ repl}\ emit\langle d\rangle$.

5: Suppose that

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

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 2, where a message is consumed and turned into a message on a continuation channel. The message can be either $emit\langle m\rangle$ or a copy of $emit\langle d\rangle$ unfolded from W. We treat only the first case; the second one is similar. Since x^+ does not occur in m, it follows that m cannot be a well-formed message for x. We thus have:

$$\begin{aligned}
Q \equiv\;& \mathcal{E}nv_{x^\pm}\,[\,R_x\{\}\,[\kappa\langle m\rangle]\,] \\
\succeq\;& \mathcal{E}nv_{x^\pm}\,[\,R_x\{\}\,[F_{m,x}] \mid emit\langle m\rangle\,] \\
\succeq\;& \mathcal{E}nv_{x^\pm}\,\left[\,R_x\{m\}\left[F'_{m,x}\right] \mid emit\langle m\rangle\,\right] \\
\asymp\;& \mathcal{E}nv_{x^\pm}\,[\,R_x\{m\}\,[\mathsf{0}] \mid emit\langle m\rangle\,] \\
\asymp\;& \mathcal{E}nv_{x^\pm}\,[\,R_x \mid emit\langle m\rangle\,]
\end{aligned}$$

by successively applying Lemmas 2, 4 (first part), 5 (second case), and 8.

6: We prove that if $R_x\{\widetilde{u}\}\,[0] \in \mathbf{R}$, x^- does not occur in m, and

$$\mathcal{E}nv_{x^\pm}\,[\, R_x\{\widetilde{u}\}\,[0] \mid emit\langle m\rangle \,] \rightarrow Q$$

then

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

for some $R' \in \mathbf{R}$ and some process T such that x^- does not occur in T. As in the previous argument, the reduction step can be only the first one of Lemma 2, and the message can be either $emit\langle m\rangle$ or a copy of $emit\langle d\rangle$ unfolded from W. We detail only the case of $emit\langle m\rangle$; that of $emit\langle d\rangle$ is similar. Let $Q' = \mathcal{E}nv_{x^\pm}\,[\, R_x\{\widetilde{u}\}\,[F_{m,x}] \mid emit\langle m\rangle \,]$. Applying Lemma 2, we obtain $Q \succeq Q'$. If m is in $\widetilde{u}$, then Lemma 4 (second part) yields

$$Q' \succeq \mathcal{E}nv_{x^\pm}\,[\, R_x\{\widetilde{u}\}\,[0] \mid emit\langle m\rangle \,]$$

so we can take $R' = R_x\{\widetilde{u}\}\,[0]$ and $T = emit\langle m\rangle$. Otherwise, let $Q'' = \mathcal{E}nv_{x^\pm}\left[\, R_x\{\widetilde{u}, m\}\left[F'_{m,x}\right] \mid emit\langle m\rangle \,\right]$. Applying Lemma 4 (first part), we obtain $Q' \succeq Q''$. In turn, applying Lemma 5, we obtain

$$Q'' \succeq \mathcal{E}nv_{x^\pm}\,[\, R_x\{\widetilde{u}, m\}\,[0] \mid x^\circ\langle\widetilde{v}\rangle \mid emit\langle m\rangle \,]$$

or

$$Q'' \succeq \mathcal{E}nv_{x^\pm}\,[\, R_x\{\widetilde{u}, m\}\,[0] \mid emit\langle m\rangle \,]$$

depending on whether m is well-formed. So we can take $R' = R_x\{\widetilde{u}, m\}\,[0]$ and $T = x^\circ\langle\widetilde{v}\rangle \mid emit\langle m\rangle$ or $T = emit\langle m\rangle$. □

7 Conclusion

In this paper, we apply a process calculus and its theory to address correctness issues for basic cryptographic protocols. The protocols are intended to convey securely a single datum, and they are described with only an abstract view of networking and cryptographic algorithms. Nevertheless, these protocols are crucial for translations that map arbitrary join-calculus processes to lower-level processes that communicate over a public network. The security of the translations depends on the correctness of the underlying protocols. We present a few sufficient conditions for correctness, and an example of a correct protocol.

This paper is a partial review and a continuation of an ongoing research project. We are currently investigating extensions of the join-calculus with constructs for authentication. The corresponding protocols implement authentication using digital signatures. They address some of the inefficiencies of the protocols discussed in this paper; for example, the presence of identity information removes the mutual anonymity of emitters and receivers, and enables some reuse of keys. So far, in this study of authentication, we have treated protocols on a case by case basis. However, we hope to obtain general correctness criteria analogous to the ones of this paper.

References

1. Martín Abadi. Protection in programming-language translations. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, pages 868–883, July 1998.
2. Martín Abadi, Cédric Fournet, and Georges Gonthier. Secure implementation of channel abstractions. Manuscript, on the Web at http://join.inria.fr/; subsumes [3] and [4].
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 and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1), January 1999. An extended version appeared as Digital Equipment Corporation Systems Research Center report No. 149, January 1998.
6. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 419–428, May 1998.
7. Andrew D. Birrell. Secure communication using remote procedure calls. *ACM Transactions on Computer Systems*, 3(1):1–14, February 1985.
8. Dominique Bolignano. Towards the formal verification of electronic commerce protocols. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 133–146, 1997.
9. Rocco De Nicola and Matthew C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
10. Cédric Fournet. *The Join-Calculus: a Calculus for Distributed Mobile Programming*. PhD thesis, Ecole Polytechnique, Palaiseau, November 1998.
11. Cédric Fournet and Georges Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of POPL '96*, pages 372–385. ACM, January 1996.
12. Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of the 7th International Conference on Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421. Springer-Verlag, August 1996.
13. Cédric Fournet, Cosimo Laneve, Luc Maranget, and Didier Rémy. Implicit typing à la ML for the join-calculus. In Antoni Mazurkiewicz and Jòzef Winkowski, editors, *Proceedings of the 8th International Conference on Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, pages 196–212. Springer-Verlag, July 1997.
14. Cédric Fournet and Luc Maranget. The join-calculus language (version 1.03). Source distribution and documentation available from http://join.inria.fr/, June 1997.
15. Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL protocol: Version 3.0. Available at http://home.netscape.com/eng/ssl3/draft302.txt, November 1996.

16. Dieter Gollmann. What do we mean by entity authentication? In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 46–54, May 1996.
17. D. Harkins and D. Carrel. RFC 2409: The Internet Key Exchange (IKE). Available at ftp://ftp.isi.edu/in-notes/rfc2409.txt, November 1998.
18. 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.
19. Gavin Lowe. Some new attacks upon security protocols. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, 1996.
20. Gavin Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 31–43, 1997.
21. Nancy Lynch. I/O automaton models and proofs of shared-key communications systems. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 14–29, 1999.
22. Catherine Meadows. Analysis of the Internet Key Exchange protocol using the NRL protocol analyzer. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, May 1999.
23. Catherine Meadows and Paul Syverson. A formal specification of requirements for payment transactions in the SET protocol. In *Proceedings of the Financial Cryptography Conference*, 1998.
24. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
25. Robin Milner. *Communication and Concurrency*. Prentice Hall International, 1989.
26. Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2:119–141, 1992.
27. 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.
28. J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *7th USENIX Security Symposium*, pages 201–216, 1998.
29. Lawrence Paulson. Inductive analysis of the Internet Protocol TLS. *ACM Transactions on Information and System Security*, 2(3), August 1999.
30. A. W. Roscoe. Intensional Specifications of Security Protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, pages 28–38. IEEE Computer Society Press, 1996.
31. Davide Sangiorgi and Robin Milner. The problem of "weak bisimulation up to". In W. R. Cleaveland, editor, *Proceedings of CONCUR'92*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 1992.
32. David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the Second USENIX Workshop on Electronic Commerce Proceedings*, pages 29–40, November 1996. A revised version is available at http://www.cs.berkeley.edu/˜daw/me.html.
33. Tatu Ylönen. SSH—Secure login connections over the Internet. In *Proceedings of the Sixth USENIX Security Symposium*, pages 37–42, July 1996.
34. Tatu Ylönen. Private communication. 1999.